KU LEUVEN

FACULTY OF ENGINEERING SCIENCE

BPE-knockout: Systematisch overzicht van BPE-tokenisers en hun gebreken met toepassing in Nederlandse morfologie

Thomas Bauwens

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdoptie Artificiële intelligentie

Promotor:
Prof. dr. Bettina Berendt

Evaluatoren:
Drs. ing. Pieter Delobelle
Drs. ir. Thomas Winters

Begeleider:
Drs. ing. Pieter Delobelle

Academiejaar 2022 – 2023

BPE-knockout: Systematisch overzicht van BPE-tokenisers en hun gebreken met toepassing in Nederlandse morfologie

Thomas Bauwens

1 juli 2023

Opgedragen aan Maurice Bauwens en Jos Deboutte,
de dokter en ingenieur wiens intelligentie ik erfde,
maar die ze als grootvaders nooit hebben zien bloeien.

Voorwoord

Evolutie van het onderwerp

De titel van het oorspronkelijk voorgestelde thesisonderzoek was

> Eng.: *Learning What To Learn: Generating Language Lessons With* BERT
> Ned.: *Leren wat te leren: taallessen genereren met* BERT

waarbij het opzet was om automatisch een opeenvolging van steeds complexere vaardigheden te bepalen die de spreker van een gegeven taal dient te beheersen (van simpele woordenschat tot complexe grammatica), om dan voor elke vaardigheid een groepje voorbeeldzinnen op te stellen die konden helpen met het leren daarvan.

Toen ik echter in de literatuur dolf, ontdekte ik dat Zanetti (2020) in haar masterthesis een zeer gelijkaardig probleem had opgelost. Bovendien bleek Tack (2021) in haar doctoraatsthesis vast te stellen dat er geen "objectief moeilijke woordenschat" bestaat, wat een ordening van taallessen zou bemoeilijken. Zanetti ondervond hetzelfde:

> *[...] what separates one level to the next is the topics students are expected to be able to discuss [...] more complex or less frequent words can be comprehended earlier if they are needed to talk about a subject.*

Het groeperen (m.a.w. clusteren) van BERT-embeddings leek me niettemin een interessante onderzoeksrichting op zich. Sia e.a. (2020) gaven reeds een aanzet o.b.v. woordtypes, maar de voor de hand liggende follow-up met tokens werd reeds door Thompson en Mimno (2020) uitgewerkt. Hun bespreking bevatte onder andere een analyse van de anisotropie van embeddings (i.e. dat hun richtingen niet gelijkmatig over de eenheidshyperbol verdeeld zijn), zoals eerst geobserveerd door Ethayarajh (2019); ook hiertoe werd een oplossing, met verklarend onderzoek, aangepakt in een recente masterthesis (Luo, 2021).

Ethayarajh suggereerde op het einde dat het interessant kon zijn om contextuele BERT-embeddings te distilleren naar statische embeddings. Belangrijk daarbij is het aggregeren (poolen) van verschillende embeddings, hetgeen Ács e.a. (2021) en Wang e.a. (2021) allen onderzochten (resp. van subwoorden naar woorden en van woorden naar woordgroepen). Onder geaggregeerde embeddings vallen ook samenstellingen; in het Engels is een samenstelling gewoon als woordgroepen te behandelen, maar in de andere Germaanse talen is een samenstelling één spatieloos woord, waarbij de complicatie kan optreden dat de subwoordgrenzen niet noodzakelijk met de echte deelwoordgrenzen samenvallen. Tokenisatie van samenstellingen is een intrigerend thesisonderwerp, maar Huck e.a. (2017) en de masterthesis van Zhou (2018) schreven er reeds over.

Slechte splitsing van samenstellingen is echter zelf een symptoom van een breder probleem: het courante tokenisatiealgoritme in transformers, BPE (Sennrich e.a., 2016), werd ontwikkeld voor bottom-upcompressie, en niet voor tokenisatie. Allerlei puntjes

van kritiek over BPE zijn verspreid doorheen de literatuur, en verscheidene varianten op BPE zijn voorgesteld de voorbije zes jaar, maar nergens is één overzicht te vinden. Dat is mijn onderwerp.

Vereiste voorkennis

Afgezien van het abstract en dit voorwoord is deze thesis vanuit *first principles* geschreven: alle termen en concepten die een doorsnee-masterstudent aan het departement computerwetenschappen niet aangeleerd kreeg, worden uitgelegd. Basiskennis over neurale netwerken is dus zowat de meest specifieke vereiste voorkennis. Hoewel dit een NLP-thesis is, moet de lezer vooraf niets over NLP weten om te kunnen volgen.

Dank

Met deze thesis ben ik aan het einde aanbeland van 16 jaar studietijd: 5 jaar lagere school, 6 jaar middelbare school, en 5 jaar universiteit. Daar zijn meer mensen aan te pas gekomen dan ik kan opnoemen, maar enkelen hebben een blijvende indruk gemaakt en krijgen vandaar een expliciete dankbetuiging:

- Magda Artois, juf in de lagere school, om mijn een extra twintigerjaar te schenken;
- Nele Verhelst, leerkracht Latijn en Grieks, voor haar warme ontvangst en enthousiaste lessen;
- Jeroen Filliaert, leerkracht Grieks, om de nodige sérieux edoch theatrale passie aan de dag te brengen en mij de wondere wereld van de taalanalyse, klassieke literatuur, antieke filosofie en breedsprakigheid te tonen;
- Roel Michiels, Donald Jacquemin, Hilde Eggermont, Bart Lambregs en Pedro Tytgat, leerkrachten wiskunde, om mij abstract te leren denken;
- Pieter Maes, leerkracht chemie, voor zijn lessen in zelfbeheersing;
- Dirk Vandepitte, professor mechanica, om mij samen met de andere ingenieurswetenschappers de nodige nederigheid in te boezemen;
- Jan De Spiegeleer, professor kansrekenen & statistiek, om mij de kans te geven mijn eerste LaTeX-boek te typesetten;
- Alle proffen die een cursus schreven, voor de intellectuele uitdagingen en de jaren die ik dankzij hen diep in gedachten verzonken mocht doorbrengen;
- Louis, goede vriend sinds de kleuterschool, voor de aanhoudende vriendschap;
- Mijn twee oma's, om altijd en overal paraat te staan om mijn ouders te helpen;
- Arne, mijn kleine broer, omdat het leven zonder hem nog veel eenzamer zou zijn;
- Papa, voor de doordachte locatie van ons huis;
- Mama, voor de dankloze maar broodnodige inspanning van de was en de *plats*;

Als laatste verdienen Pieter, mijn thesisadvisor, en Bettina, mijn thesispromotor, beide een vermelding. Pieter omdat hij vele uren naar mijn soms nogal overenthousiast gebrabbel moest luisteren, en Bettina omdat ze me hielp het kaf van het koren te scheiden.

– Thomas Bauwens, 1 juli 2023

Abstract

Natuurlijketaalverwerking (NLP) heeft met de komst van **subwoordtokenisatie** sterke vooruitgang geboekt: door invoertekst op te splitsen in tokens kleiner dan woorden is er minder dataspaarsheid, en kan er met NLP-modellen informatief gecommuniceerd worden over oneindig verschillende invoerwoorden – inclusief invoerwoorden die nooit eerder gezien zijn – met een eindig vocabularium van herkende eenheden. Een nauwere analyse toont echter dat de populairste subwoordtokeniser, **byte-pair encoding (BPE)**, tokens produceert die moeilijk interpreteerbaar zijn, daar ze zich bijvoorbeeld niet aan morfologische afbakeningen in een woord houden. Daardoor krijgt een NLP-model ambigue patronen te zien aan zijn invoer, waaruit het de structuur van de taal minder efficiënt kan leren.

Kritiek over BPE is echter verspreid doorheen de literatuur, zonder centralisatie. Deze thesis beoogt drie doelen: een **referentiekader** scheppen over subwoordtokenisatie en verwante onderzoeksdomeinen, de **kritieken** van BPE verzamelen en analyseren, en verzachtende **remedies** voorstellen.
Het eerste doel vervul ik met een uitgebreide literatuurstudie, het tweede doel vervul ik aan de hand van zowel theoretische argumentering als verifiërende experimenten in het Nederlands, en tot slot stel ik een semi-gesuperviseerd algoritme voor, **BPE-knockout**, dat het vocabularium van een reeds getrainde BPE-tokeniser trimt op zo'n manier dat er minder morfeemgrenzen binnenin de tokens van elk woord voorkomen en meer ertussen. Ook experimenteer ik met alternatieve methoden dan hiërarchische agglomeratie om een bestaand subwoordvocabularium te gebruiken voor subwoordtokenisatie.

De probleemanalyse toont dat BPE met enkele fundamentele gebreken kampt, waaronder de onmogelijkheid om tegelijk kleine morfemen en grote woordstammen te herkennen, massaal dataverlies, en productie van het ene morfeem wanneer eigenlijk een ander morfemen aanwezig is. BPE-knockout toont zichzelf superieur op de binaireclassificatietaak van het voorspellen van morfologische splitsingspunten.

De resultaten zijn een sterke indicatie dat BPE, van oorsprong een compressiealgoritme, een fundamenteel zwakke schakel is in hedendaagse NLP-modellen en niet geschikt is als de facto standaard voor subwoordtokenisatie in de toekomst. BPE-knockout biedt anderzijds een eenvoudige manier om modellen die reeds getraind zijn met BPE-tokenisers, te hergebruiken, mits enkele lichte aanpassingen in de tokeniser.

Inhoudstafel

Lijsten van illustraties

Figurenlijst

Tabellenlijst

Algoritmenlijst

Deel I

Achtergrond

H1 Inleiding

Natuurlijketaalverwerking (Eng.: *natural language processing, NLP*), het onderzoeksdomein binnen *artificiële intelligentie (AI)* dat zich ontfermt over het machinaal interpreteren van menselijke taal, is hipper dan ooit tevoren. In het bijzonder zijn de openbare testfases ("*bèta's*") van het NLP-gerichte bedrijf OpenAI niet meer uit het nieuws weg te slaan: midden 2022 lanceerde het DALL·E 2 (afgeleid van Ramesh e.a., 2022), een taalmodel dat een bijschriftzin kan omvormen tot een afbeelding, en eind 2022 vergaarde het conversationele taalmodel ChatGPT (afgeleid van Ouyang e.a., 2022) al snel wereldfaam. Beide zijn ze gebaseerd op een vorm van GPT-3 (Brown e.a., 2020), een tekstvervolediger die op zijn beurt is afgeleid van de revolutionaire transformerarchitectuur (Vaswani e.a., 2017). Meer daarover in Hoofdstuk 2.

De resultaten van dergelijke modellen zijn zo frappant dat ze ons blind kunnen maken voor gebrekkigheden onder de motorkap. We mogen niet vergeten dat GPT-3 zo'n 175 miljard af te stellen parameters bevat en naar schatting gemakkelijk enkele miljoenen dollars kostte om te trainen (Metz, 2020): dergelijke scenario's tonen dat een transformer met genoeg data, parameters en trainingtijd/-energie/-financiering kennelijk goed kan leren omgaan met al zijn onderdelen, maar dat betekent niet dat we ons tevreden moeten stellen met die onderdelen.

Laat ons eerst de eerste stap in het taalverwerkingsproces motiveren: de segmentatie van tekst. Een overzicht van de rest van de thesis komt op het einde aan bod.

1.1 Tekstvoorstellingen

Hoe kan een computer geschreven mensentaal verwerken? Noodzakelijkerwijs moet hij tekst numeriek kunnen voorstellen vooraleer hij er berekeningen op kan uitvoeren.

Voor alledaagse toepassingen (bv. schrijfapplicaties, webbrowsers ...) stelt men karakters van een digitale tekst (een *string*) voor met een *codering* (Eng.: *encoding*) die hun elk een unieke rij van bits toewijst. In de UTF-8-codering wordt de string *"Heb geen ὕβρις!"*:

01001000 01100101 01100010 00100000 01100111 01100101 01100101 01101110
00100000 11100001 10111101 10010101 11001110 10110010 11001111 10000001
11001110 10111001 11001111 10000010 00100001

waarbij de spaties enkel voor de leesbaarheid zijn toegevoegd. We zouden karakters dus als rij van 1'en en 0'en kunnen verwerken; echter, de bovenstaande codering maakt

de invoer minstens achtmaal zo lang als de string, en is bovendien arbitrair – of een bepaalde bit nu aan of uit staat, heeft geen betrekking tot hoe een karakter zich gedraagt. Bovendien zijn de Griekse karakters twee tot drie keer zo lang als de Latijnse.

We zouden evengoed elke 8 bits (1 byte) als decimaal getal kunnen interpreteren; zonder teken, en met meest significante bit eerst, krijgen we de rij

72 101 98 32 103 101 101 110 32 225 189 149 206 178 207 129 206 185 207 130 33.

Dat is ongeveer even lang als de string zelf, maar net zoals de codering is deze interpretatie van de bitstring arbitrair. Het is aldus een slecht idee om rechtstreeks bewerkingen op betekenisloze getallen uit te voeren.

De oplossing is om dergelijke volgnummers (*id*'s) niet rechtstreeks naar een model te sturen, maar om er een *look-up table (LUT)* tussen te zetten die de id's omzet naar zinnig bewerkbare operanden. Zoals in de rest van machine learning zijn die operanden reële vectoren; meer over die ontwerpkeuze in §2.3. Omdat karakters op die manier "ingebed" worden in een ruimte met een zekere semantiek, heten die vectoren *embeddings*.

1.2 Tokenisatie

Een laatste vrijheidsgraad is de teksteenheid die we coderen. Het is niet evident dat we taal zouden verwerken als karakters. Veel kinderen leren lezen met de zogenoemde *whole-word method*, waarbij ze woorden in hun geheel herkennen i.p.v. als letters; als we taal willen begrijpen, dan lijkt het inderdaad beter om op grover niveau te rekenen.[1]

Even wat terminologie: elke string – karakter, woord ... – waaraan we een id toewijzen heet algemeen een *token*. De module die tekst opsplitst in tokens heet de *tokeniser*. Alle tokens met dezelfde id zijn van hetzelfde *type* (Zhou, 2018): stel dat we tekst op basis van woorden opsplitsen, dan bevat de string *"het is wat het is"* 5 woordtokens, maar slechts 3 woordtypes. De afbeelding van types op id's heet het *vocabularium* V.

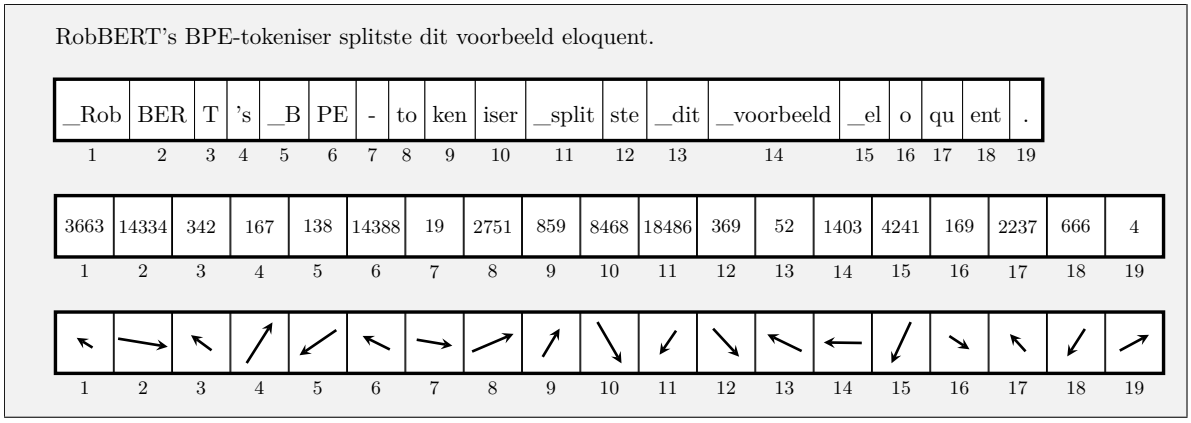
Een extreem belangrijk gevolg van de omzetting van tokens naar id's (en van id's naar embeddings) is dat de module die op de tokeniser volgt (een taal- of vertaalmodel, zie §2.2) *volledig blind is voor de invoertekst zelf, alsook voor de karakters binnen een token* (Clark e.a., 2022). De uitvoer van de tokeniser is een rij van natuurlijke getallen die elk een string met arbitraire lengte voorstellen. Een observator van die rij is zich compleet onbewust van geschreven tekst; men kan de woorden *"hond"* en *"honden"* niet beter van elkaar onderscheiden dan *"hond"* en *"computer"*. Figuur 1.1 toont wat elke component die voorafgaat aan het eigenlijk model te zien of niet te zien krijgt.

1.3 Trammelant in tokenisatieland

1.3.1 Leestekens

Tokenisatie in woorden is moeilijker dan splitsen op spaties. Leestekens plakken meestal aan een woord (behalve in het Frans), maar toch zijn ze er geen deel van. In het geval van de Saksische genitief ("bezits-*s*") houdt het anderzijds steek om de apostrof niet compleet te isoleren (*papa_'s_job* i.p.v. *papa_'_s_job*). In het Engels splitst men woordcontracties als *don't* en *isn't* best niet op de leestekens, maar liever als *do_n't* en

[1] Net als in NLP-onderzoek is er in de pedagogie echter consensus dat de wholewordmethode voor kansarmoede zorgt. Ik verwijs de lezer naar het boek *Why Johnny Can't Read* en §1.4.



Figuur 1.1 – Dataformaten aan de interfaces in een NLP-systeem. Van boven naar onder: wat de tokeniser ziet (karakters), wat het vocabularium ziet (subwoorden), wat de embedder-LUT ziet (token-id's), en wat een neuraal model ziet (embeddingvectoren).

is_n't (Mielke e.a., 2021). Een *pre-tokeniser* dient om dergelijke regeltjes[2] toe te passen. Er is weinig innovatie mogelijk in pre-tokenisers, waardoor moderne neurale systemen als XLM (Lample en Conneau, 2019) de pre-tokeniser van oude statistische systemen hergebruiken (in het geval van XLM bv. Moses).

1.3.2 Open vocabularium

Het gebruik van woordtokens i.p.v. karaktertokens zorgt echter voor een veel fundamenteler probleem: er zijn oneindig woorden te vormen – de taal heeft een *open vocabularium* (Sennrich e.a., 2016) – i.t.t. ons eindige alfabet. Zelfs als taalverwerking zich beperkt tot geldige woorden, zijn er nog oneindig: men kan altijd onvoorspelbare eigennamen, leenwoorden en neologismen verzinnen. In alle Germaanse talen (buiten het Engels) zijn er zelfs oneindig woorden zónder die categorieën, vermits ze *gesloten samenstellingen* toelaten, i.e. vorming van nieuwe woorden door bestaande woorden aaneen te schrijven. Enkele Nederlandse voorbeelden:

universiteitsbibliotheek, *fouriertransformatie*, *Queenalbum*,
geitenwollensokkendrager, *muggenziften*, *dodehoekongeval*, *groentesoep*

Hoewel de verzameling van alle geldige woorden van de taal telbaar is (want de verzameling van alle strings met eindig alfabet is dat reeds), heeft het weinig zin om elk een id toe te wijzen. V in het geheugen of in opslag bewaren zou onmogelijk zijn. Zelfs als het dat wel was, dan nog zou het merendeel nooit geraadpleegd worden, want de meeste samenstellingen komen spaars/nooit voor (bv. *keukentafeluniversiteitsauto*). Vandaar dat men open vocabularia "sluit" door ze af te snijden bij een $|V| = \tau < \infty$.

Woorden die niet in V zitten, noemt men *out-of-vocabulary (OOV)*. De tokeniser vervangt zulke woorden in de invoer met een speciaal token ⟨`UNK`⟩ in zijn uitvoer, dat zoveel betekent als "hier stond een woord waar het vocabularium geen apart nummer voor heeft, dus verzin maar iets", waarbij overige informatie verloren gaat.

Kan de tokeniser OOV-samenstellingen redden door ze te herleiden tot een rij van deelwoorden die wel in V zitten? Zelfs als we een betrouwbare samenstellingsdetector hadden, zou splitsen op zich niet volstaan:

[2] In het Engels is de regel voor de Saksische genitief eenvoudig, gezien die altijd een apostrof heeft. In het Nederlands is het moeilijker zo'n regel op te stellen, gezien de meervouds-*s* ook met apostrof voorkomt (na klinkers) en de bezits-*s* ook zonder apostrof voorkomt (na medeklinkers).

- *Compositionele* samenstellingen kunnen verschuiven van betekenis:
  het rode wijnglas ≠ *het rodewijnglas*, en *de gemeentehuizen* ≠ *de gemeente huizen*.
- *Non-compositionele* samenstellingen – ook wel *lexicaliseringen* (Shwartz en Waterson, 2018) – verliezen hun betekenis volledig:
  muggenziften ≠ *muggen ziften*, en *boterham* ≠ *boter ham*.
- *Interfices* zijn lijmpartikels die ontstaan wanneer deelwoorden samen worden geplaatst. Alle Germaanse talen hebben interfices, en in alle zijn ze notoir om hun onvoorspelbaarheid. Bijvoorbeeld:

  Ned.: *schoonheidsideaal*, *schapenwol*, *eierschaal*

  Dui.: *Arbeitstier*, *Herzensgüte*, *Geisterfahrer*

  Merk daarbij dat de talen elkaar niet volgen in of er een interfix is (vgl. *spookrijder*; *Schafwolle*). Toch hebben mensen met een Germaanse moedertaal weinig moeite om intuïtief de correcte interfix te gebruiken; uit empirisch onderzoek van hun gedrag o.b.v. fictieve voorbeelden zijn sommige patronen af te leiden (Krott, 2001).

  De tokeniser mag een interfix uiteraard niet als apart woord afsplitsen, noch aan het linkerdeelwoord laten hangen. Weggooien is anderzijds niet altijd gerechtvaardigd: een *boerenzoon* is de zoon van één boer (*boer* + *zoon*), maar een *boerenbond* is een bond van meerdere boeren (*boeren* + *bond*).
- *Klankvervorming* doet zich soms voor wanneer deelwoorden samen worden geplaatst: *schip* + *vaart* = *scheepvaart*, terwijl *scheep* zelf geen woord is.

Hoewel woordgebaseerd splitsen voor sommige toepassingen volstaat (zie § 2.4.1), heeft men voor taalbegrip (Eng.: *natural language understanding, NLU*) en taalgeneratie (Eng.: *natural language generation, NLG*) een intelligentere tokeniser nodig die minder verliesloos optreedt met eindige $|V|$, en alsnog een open vocabularium nabootst.

1.3.3 Onderlinge invloed van woordtypes

De *distributionele hypothese (DH)* stelt het volgende (Sahlgren, 2008):

> **Distributionele hypothese (DH)**
> De betekenis van een woordtype wordt bepaald door de contexten waarin tokens van dat type voorkomen.

Woorden met gelijkaardige betekenis komen zo in gelijkaardige contexten voor. De datasets (*corpora*) die men gebruikt om machines taal aan te leren, bestaan zo ook uit voorbeeldcontexten (zinnen, paragrafen ...) i.p.v. losse woorden. Volgens de DH wordt de geleerde betekenis van een woordtype des te helderder des te meer gebruiksvoorbeelden er voor zijn.

Om meer voorbeelden te verkrijgen voor een bepaald woordtype zouden we, volgens de DH, die van woordtypes met gelijkaardige betekenis kunnen hergebruiken. Dat is een cirkelredenering voor synoniemen, want zonder genoeg data kunnen we niet weten dat bv. *carnivoor* en *vleeseter* exact hetzelfde betekenen. Er zijn echter niet-synonieme woorden met gelijkaardige betekenis die we zonder data kunnen verbinden, nl. op basis van gelijkaardige morfologie: *hond* en *honden* zijn twee woordtypes die bij eenzelfde basisvorm HOND horen, net zoals *speel*, *speelt*, *spelend* en *gespeeld* allemaal bij SPELEN horen. Dergelijke typefamilies heten samen een *lexeem* (Carstairs-McCarthy, 2008), en hun basis (HOND, SPELEN ...) heet een *lemma*. Een woordenboek lijst lemmata op.

Wat terminologie rond lexeemvorming (Ataman en Federico, 2018):

- *Flexie* (Eng.: *inflection*): vorming van woordtypes vanuit een lemma (m.a.w. vorming van de leden van een lexeem). Flexie van naamwoorden heet *verbuiging* (Eng.: *declension*), flexie van werkwoorden heet *vervoeging* (Eng.: *conjugation*).
- Woordformatie: vorming van nieuwe lexemen. Een lexeem vormen o.b.v. één ander lexeem (bv. *schoon* → *schoonheid*) heet *afleiding* (Eng.: *derivation*).[3] Een lexeem vormen door samenvoeging van twee andere heet *samenstelling* (Eng.: *compounding*).

Talen worden vaak gecategoriseerd m.b.t. hun hoeveelheid flexie. Frans heeft extreem flexionele vervoegingen, Duits heeft naamvallen en dus zeer flexionele verbuigingen, en Latijn en Grieks zijn extreem in beide. Nederlands ligt tussen Engels en Duits.[4] Alle voorgenoemde talen zijn *fusioneel*, d.w.z. dat alle lexicale eigenschappen (bv. naamval, genus, getal, wijs, tijd ...) in één affix wordt verpakt. *Agglutinatieve* talen, in het bijzonder Turks, plakken per eigenschap een extra affix aan het woord.

Alle woordtypes van een lexeem zullen duidelijk in gelijkaardige contexten voorkomen, modulo hun lexicale eigenschappen. Het is dus een goed idee om al hun data te poolen en hun betekenis gezamenlijk te leren. Sterker nog: hoe groter een lexeem is, hoe noodzakelijker het is om woordtypes elkaar te laten beïnvloeden. In het Frans heeft een werkwoordlexeem makkelijk één woordtype per wijs (indicatief, subjunctief, conditioneel), tijd (heden, verleden, toekomst), persoon (1, 2, 3) en aantal (enkelvoud, meervoud). Enkele vormen van het werkwoord CHOISIR ("kiezen"):

ind. pr.: *choisis, choisit, choisissons, choisissez, choisissent*
1. sing.: *choisirai, choisissais, choisirais, ai choisi(e)(s), choississe ...*

Vergelijk dat met het Engelse CHOOSE:

ind. pr.: *choose, chooses*
1. sing.: *chose, have chosen.*

In §1.2 bleek dat de karakters binnen een woordtype verborgen worden achter één abstract getal door een woordtokeniser. Dat wil zeggen dat indien we hetzelfde corpus in het Frans en in het Engels hebben, de woordtypes in het lexeem van CHOISIR elk gemiddeld véél minder voorbeelden krijgen dan die van CHOOSE, terwijl het concept "kiezen" in beide corpora exact evenveel keer wordt geïllustreerd!

Daar komt nog bij dat V veel sneller opgevuld raakt voor het Frans en er dus minder concepten te leren zijn. Anderzijds moet elk woordtype letterlijk in de dataset aanwezig zijn opdat het aan V kan worden toegevoegd, wat dan weer betekent dat ongeziene vervoegde vormen – die achteraf alsnog evenveel kans hebben om voor te komen als geziene vervoegde vormen – door ⟨UNK⟩ vervangen worden door de tokeniser.

1.4 Subwoorden

De twee bovenstaande problemen – informatieverlies door ⟨UNK⟩ vanwege het open vocabularium, en dataspaarsheid vanwege grote lexemen – zijn op te lossen door te tokeniseren met eenheden die kleiner kunnen zijn dan woorden maar groter kunnen zijn dan

[3] Afleiding gebeurt net als flexie vaak met een *affix* (partikel dat aan een woordstam wordt geplakt), maar bij flexie ontstaat er dan geen nieuw lexeem.

[4] Het Nederlands kent geen naamvallen en geen enorm rijke werkwoordsvervoeging, maar onze klanken veranderen nogal gemakkelijk. Zo wordt *"een aap eet een banaan"* in het meervoud *"apen eten bananen"*, waarbij alle woordstammen veranderen. Medeklinkers kunnen verdubbelen (*"kat"* → *"katten"*) of niet (*"bad"* → *"bāden"*), vervormen (*"poes"* → *"poezen"*) etc ...

karakters: *subwoorden* (Eng.: *subwords*). Analoog aan de terminologie voor woordgebaseerde tokenisers is V een verzameling *subwoordtypes* die zich uiten als *subwoordtokens*.

1.4.1 "How it started..."

Karakters zijn een deelverzameling van subwoorden. Door V te initialiseren als de verzameling van alle karakters, is eender welk woord op zijn minst te tokeniseren als de rij van zijn karakters, zelfs als het woord nog nooit in de dataset gezien werd: er treedt geen informatieverlies op, en er is geen nood aan ⟨UNK⟩.

Anderzijds kunnen subwoorden de unieke onderdelen van een lexeem scheiden van de generieke. Laat ons daartoe de Franse vervoeging van FINIR ("eindigen") bekijken:

ind. pr.: *finis, finit, finissons, finissez, finissent*
1. sing.: *finirai, finissais, finirais, ai fini(e)(s), finisse* ...

Vergeleken met de vervoeging van CHOISIR hierboven zijn de suffices duidelijk hetzelfde en is alleen de woordstam anders. Een subwoordvocabularium als

$$V = \{..., \text{fin}, \text{chois}, \text{-ir}, \text{-is}, \text{-it}, \text{-issons}, \text{-issez}, \text{-issent}, \text{-irai}, \text{-issais}, \text{-irais}, \text{-i}, \text{-isse} ...\}$$

kan daar duidelijk veel beter mee omgaan dan een woordvocabularium. $|V|$ groeit nu met welgeteld 1 subwoord telkens we een werkwoord van dezelfde aard tegenkomen (GRANDIR, RÉFLÉCHIR ...), i.p.v. met alle woordtypes van het lexeem.

Het spaarsheidsprobleem is daarmee ook opgelost: elk woord in het lexeem van CHOISIR draagt nu bij tot het begrijpen van het concept, voorgesteld door hetzelfde subwoord "chois". Nog beter: er is niet alleen invloed tussen de woorden binnen hetzelfde lexeem, maar ook tussen woorden van andere lexemen. Contexten waarin vervoegingen van FINIR voorkomen, dragen bij tot het begrijpen van die van CHOISIR omdat de tokeniser dezelfde suffices vindt.

Dat laatste helpt ook om uit onbekende woorden toch zoveel mogelijk interpretatie te halen. Eerder bleek al dat een ⟨UNK⟩ een woord volledig verzweeg, dus zelfs als het grammaticaal grotendeels herkenbaar was. Stel dat het lexeem van ACCOMPLIR volledig onbekend is voor V, dan zal de tokenisatie van bv. *accomplissait* alsnog een subwoord *-issait* opleveren (en de rest wordt al dan niet volledig in karakters gesplitst), waardoor het model na de tokeniser al zijn kennis over de uitgang *-issait* wel nog kan toepassen. Ook menselijke lezers achterhalen de betekenis van onbekende woorden o.b.v. de herkenbare stukken (*morfemen*) waaruit ze bestaan (Sennrich e.a., 2016).

1.4.2 "How it's going..."

Dat zijn allemaal mooie idealen. Jammer genoeg houden subwoordtokenisers zich in de praktijk zelden aan morfeemgrenzen. Het populairste tokenisatiealgoritme, BPE – waar deze thesis nog uitvoerig op zal ingaan – is volledig gebaseerd op voorvalsstatistieken van karaktersequenties, dus worden morfeemgrenzen vaak niet gerespecteerd. Tabel 1.1 toont enkele voorbeelden uit de literatuur waarin BPE-tokenisatie morfemen opsplitst en/of de resulterende delen van aangrenzende morfemen samenvoegt.

Het kan niet anders dan dat dergelijke misbaksels voor verwarring zorgen bij het NLP-model dat na de tokeniser volgt. Woordvorming gebeurt in alle talen met welbepaalde regels die welbepaalde atomen samenplaatsen. In Germaanse talen zijn die atomen morfemen. De atomen en samenplaatsingsregels bieden de betrouwbaarste letterpatronen in de taal; eender welke foutieve identificatie van atomen (bv. doen alsof twee halve morfemen samen een atoom zijn) is geen onderdeel van de regelmatige structuur van de

	BPE	ideaal
Sennrich e.a. (2016)	Gesundheits|forsch|ungsin|stitute	Gesund|heit|s|forschung|s|institute
Ataman e.a. (2017)	kan|unda (kan = bloed)	kanun|da (kanun = wet)
	ağ|lamayacak (ağ = net)	ağlama|yacak (ağlamak = huilen)
Bostrom e.a. (2020)	Comple|t|ely	Complete|ly
	t|ric|y|cles	tri|cycle|s
	n|an|ote|chn|ology	nano|techno|logy
He e.a. (2020)	vul|n|era|bility	vulner|ability
	t|ighter	tight|er
	emb|arked	embark|ed
	predic|table	predict|able
Vilar e.a. (2021)	bewer|t|ungsin|stru|mente	bewert|ung|s|instrumente
	gefan|gen|genommen	ge|fangen|ge|nommen
	verbrau|ch|spru|ef|standard	verbrauch|s|pruef|standard
	haushalt|war|enab|teilung	haushalt|waren|abteilung
	not|arz|tau|tos	notarzt|auto|s

Tabel 1.1 – Voorbeelden in de literatuur die amorfologische tokenisatie met BPE tonen in het Duits, Turks en Engels. In het rood: frappante tokens, bv. bestaand uit twee halve morfemen.

taal, en is dus geen consistent patroon doorheen een corpus.[5] Een significant deel van deze thesis rust op de waarheid van die *morfologische hypothese (MH)*:

> **Morfologische hypothese (MH)**
> NLP-modellen met subwoordtokenisatie presteren beter wanneer subwoordgrenzen overeenkomen met morfologische grenzen.

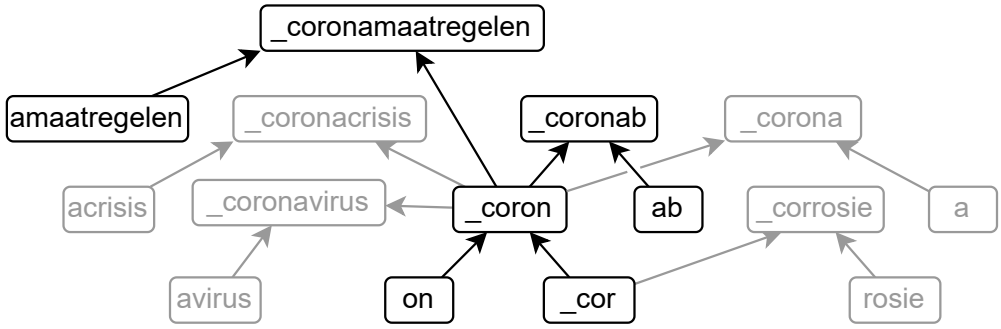
Uiteraard kan een neuraal model met arbitrair veel vrijheidsgraden en trainingdata leren omgaan met suboptimale invoer, maar zulke middelen zijn onrealistisch of onwenselijk. In elk geval schiet er nu weinig over van het mooie verhaal over subwoorden in de vorige sectie, en wie weet is het zelfs voordeliger om alleen met karakters te werken (cfr. §2.5.4).

Daar stoppen de problemen niet. Figuur 1.2 toont een voorbeeld van hoe subwoorden worden opgebouwd en vervolgens gebruikt met het BPE-algoritme. Veronderstel voorlopig[6] dat een BPE-tokeniser probeert om een gegeven string voor te stellen met subwoordtokens die zo hoog mogelijk in de graaf liggen.
Gegeven de string *"coronamaatregelen"* stellen we twee dingen vast o.b.v. de figuur:

- De samenstelling *coronamaatregelen* kwam in de trainingdata blijkbaar genoeg voor dat het op zich een subwoord in V is geworden. Dat betekent dat het door te tokeniser achter één id wordt geanonimiseerd, en er dus geen kennisoverdracht is tussen andere samenstellingen met *corona* (bv. *coronavirus*) en *-maatregelen* (bv. *beleidsmaatregelen*).
- Zelfs áls we dat subwoord zouden vergeten, dan nog vervallen de theoretische voordelen van subwoordvocabularia, want er is veel minder consistente informatie over het gebruik van het nonsenssubwoord *-amaatregelen*.

[5] Morfemen laten bovendien simpele maar effectieve afbeeldingen toe: ze zijn vaak 1:1 af te beelden op woordsoorten (*-ing* en *-heid* maken meestal substantieven), en in vertalingstoepassingen is er vaak een 1:1-equivalent morfeem (Huck e.a., 2017). Vergelijk *opbouw*, *buildup*, *Aufbau* ...

[6] De exacte werking komt nog, maar in dit voorbeeld volstaat het om te vereenvoudigen.



Figuur 1.2 – Totstandkoming van enkele BPE-subwoordtypes, o.a. *coronamaatregelen*. Bron: gecorrigeerde[7] figuur 1 van Delobelle e.a. (2022) met enkele uitbreidingen in het grijs.

Indien u het gevoel krijgt dat BPE zich eerder als een compressiealgoritme dan een tokeniser gedraagt, heeft u het goed (cfr. § 2.1.2); in zekere zin is het een voordeel dat het model na de tokeniser een kortere rij van id's toegestuurd krijgt, maar er zijn duidelijk grenzen. Wederom concluderen we dat indien men niet voorzichtig omgaat met subwoorden, de voordelen t.o.v. karakters niet kunnen opboksen tegen de nadelen.

Om deze vermanende sectie af te sluiten, wil ik nog belichten dat de theorie van subwoordtokenisatie een onderliggende aanname maakt die lijkt op de DH, maar niet altijd juist is. Ik noem haar de *grammale hypothese (GH)*.[8]

Grammale hypothese (GH)
Strings met meer karakteroverlap hebben een meer verwante betekenis.

De GH klopt zeker voor de Franse voorbeelden hierboven: alle woorden die na tokenisatie het subwoord *"choisi"* bevatten, informeerden inderdaad allemaal over hetzelfde concept. Zo ook voor *"hond"* en *"hond|en"* (zelfde lexeem), voor *"aandacht"* en *"aandacht|ig"* (afleiding) etc ...
Desalniettemin toont een prefixanalyse (zie §4.14) dat de GH het soms laat afweten: zo zijn *"bond"* en *"bond|ig"*, *"recht"* en *"recht|s"*, *"zon"* en *"zon|de"*[9] ... alle ongerelateerd.

1.5 Samengevat

Laat ons de belangrijkste ideeën in dit hoofdstuk samenvatten.

- NLP-modellen werken niet op traditionele tekstcoderingen.
- NLP-modellen hebben zelfs geen weet van de karakters in de invoertekst, gezien er twee volledig ondoorzichtige indirecties/abstracties tussen de tekst en het model zitten: het vocabularium zet tokens met arbitraire lengte om tot natuurlijke getallen, en de embedding-LUT zet die om naar reële vectoren (zie bv. Figuur 1.3).
- Woordtokenisers kampen met oneindige vocabularia en dataspaarsheid.

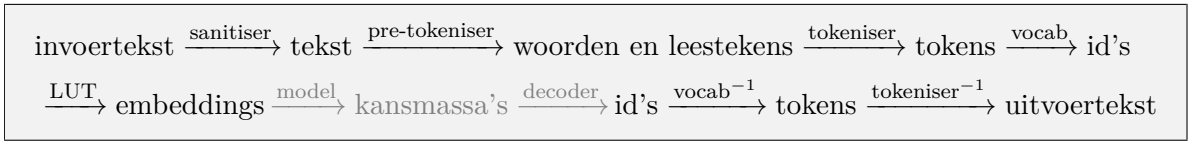
[7] De oorspronkelijke figuur is niet consistent met het tokeniserbestand op HuggingFace. Ze toont bv. de samentrekking *Coron+amaatregelen*, met hoofdletter en zonder beginkarakter, maar dat subwoord bestaat niet. Uit diens voorouder *Cor* vormt de figuur het subwoord *Cor+ona*, maar gezien de voorouder *_cor* moet zijn, heb ik dat door *_coron+a* vervangen.

[8] Aangezien we de uitspraak "gelijke betekenis volgt uit gelijke distributie" met "distributioneel" aanduiden, is "gelijke betekenis volgt uit gelijke letters (Gri.: γράμματα)" als "grammaal" te benoemen.

[9] In de betekenis van "hamartia", niet "in de zon gelegen hebben".

- Subwoordtokenisers zijn flexibel genoeg om met een eindig vocabularium een oneindig vocabularium te modelleren (bv. lexemen te modelleren zonder elk woordtype van elk lexeem expliciet op te slaan).
- Subwoordtokenisers laten in theorie toe om data van verschillende woordtypes te hergebruiken voor het leren van dezelfde morfemen.
- Subwoordtokenisers houden zich in de praktijk niet aan morfeemgrenzen.

Tot nu toe werden de mechanismen achter tokenisers, embeddings en de daaropvolgende modellen in het ongewisse gelaten. In Hoofdstuk 2 worden ze dieper toegelicht.



Figuur 1.3 – NLP-pipeline voor modellen met tekst als uitvoer.

1.6 Thesisoverzicht

Het middengedeelte van deze thesis bestaat uit twee delen:

> Deel I: Achtergrond omvat dit hoofdstuk, vervolgens een uitgebreide literatuurstudie over BPE-varianten, BPE-alternatieven, embeddings en modellen, en tot slot de onderzoeksvragen waarop ik me verder toeleg.
>
> Deel II: Contributie omvat enerzijds een verzameling van problemen met BPE die men in de literatuur opnoemt en die ik met elkaar verbind en empirisch staaf, en anderzijds enkele voorstellen voor nieuwe BPE-gebaseerde tokenisers. In de conclusie komen de kernideeën nog eens aan bod.

In Deel III: Appendices bied ik uitbreidende informatie ter ondersteuning van de rest:

> Appendix A: Notatie geeft een overzicht van onambigue symbolen voor het aangeven van aantallen in corpora onderhevig aan een tokeniser.
>
> Appendix B: Viterbialgoritmes biedt een opfrisser over viterbialgoritmes, daar die veelvuldig terugkomen in de verschillende hoofdstukken. Er worden twee toepassingen in stringsegmentatie gegeven.
>
> Appendix C: Datasets beschrijft welke datasets aangewend zijn in de experimenten binnen deze thesis en hoe de data vooraf verwerkt zijn.
>
> Appendix D: Extra figuren bevat uitvergrote figuren en tabellen.

H2 Literatuurstudie

In Hoofdstuk 1 gaf ik een draad van redeneringen die ons tot het innovatieve idee van subwoordtokenisers leidde. Dergelijke uitvindingen gebeuren niet in een vacuüm, waardoor ze worden beïnvloed door voorgaand onderzoek en zelf invloed uitoefenen op later onderzoek. Het loont dus vooreerst om subwoordtokenisatie in een bredere historische context van het voorbije decennium te plaatsen; zo zal blijken dat de geschiedenis van het attentionmechanisme parallel loopt met die van BPE, de bekendste subwoordtokeniser.

De uitvoer van een tokeniser is uiteindelijk bedoeld om tot embeddings te worden omgezet, waaruit een neuraal model vervolgens taalvaardigheid kan leren. Beide mechanismen hebben hun onderzoeksniches die ook weer sporen nalaten op tokeniseronderzoek. Vandaar geef ik voor de volledigheid een kort overzicht van embeddings in §2.3 en modellen in §2.2. Ander onderzoek over het omgaan met subwoorden volgt in §2.4.

We zagen tevens dat subwoorden in de praktijk niet zo ideaal zijn als bedoeld. Vandaar is er een heel gamma aan BPE-variëteiten verzonnen: alles daarover in §2.5.

2.1 Drie recente revoluties in NLP

2.1.1 2015: Bahdanau's attention

De geschiedenis van *neurale netwerken (NN's)* strekt zich ondertussen uit overheen meer dan een halve eeuw, en men heeft tijdens die periode veel obstakels overwonnen. Lang geleden publiceerden Hochreiter en Schmidhuber (1997) bijvoorbeeld hun paper over de *long short-term memory (LSTM)*-eenheid die, in tegenstelling tot een *feed-forward*-NN (*FFNN*), een rij van vectoren kon verwerken zonder vaste rijlengte, en in tegenstelling tot een *recurrent* NN (*RNN*) geen extreme gradiënten ondervond tijdens het trainingsproces. Zinnen in natuurlijke taal zijn eveneens zo'n invoer zonder vaste lengte, maar om grote LSTM's op taalproblemen toe te passen was het wachten op de komst van embeddings (§2.3) en vervolgens de grote datasets (*big data*) en forse hardwareverbeteringen (met name de parallellisatie in GPU's) van het laatste decennium.

Machinale vertaling (Eng.: *machine translation, MT*) gebeurde tien jaar geleden nog op basis van tabellen die telden hoe vaak een zeker woord in de brontaal werd vertaald naar een zeker ander woord in de doeltaal. Sutskever e.a. (2014) merkten op dat die *statistische MT (SMT)* kon vervangen worden door *neurale MT (NMT)* met behulp van LSTM's. Hun model, *Seq2Seq*, past een LSTM (de *encoder*) toe op de rij van

woordembeddings in de bronzin, en beschouwt de resulterende LSTM-staat als vectoriële voorstelling van de volledige zin. Die wordt dan gebruikt als initiële staat van een tweede LSTM (de *decoder*), wiens eerste invoer een dummywoord (*end-of-sentence*, ⟨EOS⟩) is. Op basis van zijn staat en invoer produceert de decoder enerzijds een nieuwe staat en anderzijds een uitvoerkansmassa over de woorden in de doeltaal. Daaruit wordt het volgende woord van de uitvoer gesampled, en diens embedding wordt als volgende *invoer* gegeven. Dat *autoregressieve* proces wordt herhaald totdat ⟨EOS⟩ gesampled wordt.

Hoewel het ontwerp van LSTM's dient om lange invoerrijen aan te kunnen, is er een fundamenteel probleem met het encoder-decodermodel van Seq2Seq: hoe langer de invoerzin, hoe meer informatie de encoder in de voorstelling van de zin probeert te proppen. Die voorstelling heeft daarentegen een vast aantal dimensies (dezelfde als een LSTM-staat), wat dan voor een flessenhals (Eng.: *information bottleneck*) zorgt; de encoder kan de decoder niet scherp informeren over de betekenis van een lange zin.

Bahdanau e.a. (2015) lossen dat op met enkele aanpassingen aan Seq2Seq. Ten eerste moet de encoder een bidirectionele LSTM (BiLSTM) gebruiken, d.w.z. een duo van LSTM's waarin de ene de bronzin voorwaarts en de andere de zin achterwaarts leest. Ten tweede moet de staat van de BiLSTM (de concatenatie van de staat die elk van beide LSTM's produceert als reactie op een invoerwoord) doorheen de tijd bijgehouden worden, i.p.v. alleen de laatste beschikbaar te stellen. Ten derde krijgt de decoder niet alleen de vorige uitvoer als invoer, maar ook een *som van de encoder-staten gewogen met hoezeer ze bij de vorige decoder-staat horen*, genaamd de *contextvector*. M.a.w. kan de decoder per iteratie aandacht (Eng.: *attention*) besteden aan de individuele woorden in de bronzin die relevant zijn voor het volgende vertaalde woord.

Het is moeilijk na te gaan wie attention heeft uitgevonden. Galassi e.a. (2021) tonen in hun overzichtspaper een tiental varianten die voortborduren op het algemene idee van "⟨combinatie⟩ van vectoren gewogen met ⟨relevantie⟩", en zij menen dat Bahdanau e.a. de eerste waren die *additieve* attention voorstelden.
Formeel ziet dat eruit als volgt. Laat ons de LSTM-iteratie in de encoder en decoder telkens aangeven met "$[t]$". Stel dat de voorbeeldzin n woorden bevat en de bijhorende BiLSTM-staten $\vec{h}_e[1] \dots \vec{h}_e[n] \in \mathbb{R}^H$ zijn. De relevantiefunctie $a : \mathbb{R}^H \times \mathbb{R}^H \to \mathbb{R}$ definiëren we als een trainbaar FFNN, en de contextvector $\vec{c}[t]$ is een gewogen som:[10]

$$a(\vec{h}_e, \vec{h}_d) = \boldsymbol{W}_o \tanh(\boldsymbol{W}_e \vec{h}_e + \boldsymbol{W}_d \vec{h}_d) \tag{2.1}$$

$$\vec{\alpha}[t] = \text{softmax}\Big(\Big[a(\vec{h}_e[1], \vec{h}_d[t-1]) \quad \dots \quad a(\vec{h}_e[n], \vec{h}_d[t-1])\Big]\Big) \tag{2.2}$$

$$\vec{c}[t] = \sum_{i=1}^{n} \alpha_i[t] \vec{h}_e[i] \tag{2.3}$$

met $\boldsymbol{W}_e, \boldsymbol{W}_d \in \mathbb{R}^{A \times H}$ en $\boldsymbol{W}_o \in \mathbb{R}^{1 \times A}$, en $\vec{h}_d[t-1]$ de vorige decoderstaat.

De bahdanau-architectuur is reeds een verbetering op Seq2Seq, maar heeft nog twee belangrijke gebreken die aanleiding hebben gegeven tot de andere twee revoluties in deze sectie. Ten eerste waren woordvocabularia nog steeds in trek anno 2015, waardoor er bv. veel eigennamen verloren gingen en door ⟨UNK⟩ vervangen werden in de tokeniser. Ten tweede erft de bahdanau-architectuur stiekem nóg een flessenhals van Seq2Seq: de staat binnenin de drie LSTM's in het systeem.

Er zijn slechts enkele manieren waarop woorden met elkaar kunnen interageren in de

[10] Ter herinnering: *softmax* transformeert een vector $\vec{z}$ naar een genormaliseerde vector met elementen $\text{softmax}(\vec{z})_i = e^{z_i} / \sum_k e^{z_k}$.

bahdanau-architectuur, terwijl die interactie juist vitaal is voor NLU en MT. Een invoerembedding "ziet" de andere invoerembeddings alleen in verwerkte vorm omdat ze samengepropt zijn in de binnenkomende LSTM-staat. Net als de contextvector kan zo'n staat niet oneindig veel informatie blijven meesleuren, met als gevolg dat het begin en het einde van lange invoerzinnen een heel troebel beeld hebben van elkaar. Voor de teruggekoppelde uitvoerembeddings geldt hetzelfde.
Die dingen waarover attention wordt gevoerd, nl. de encoderstaten $\vec{h}_e[1] \ldots \vec{h}_e[n]$, komen nooit in direct contact met elkaar; de sommatie in Vgl. 2.3 is de meeste invloed die ze op elkaar kunnen uitoefenen. Voor de decoderstaten $\vec{h}_d[1] \ldots \vec{h}_d[m]$ is de situatie nog erger, want die worden zelfs niet eens onthouden – immers, er wordt toch geen attention over gevoerd. In de encoder en de decoder van een bahdanaumodel is er dus een soort horizon waarachter woorden zich niet langer bewust zijn van elkaars aanwezigheid, en dat maakt langetermijnafhankelijkheden in de uitvoerzin moeilijk bij te houden.

Niettemin was deze architectuur een groot succes. Ze was weliswaar achterhaald vanaf de publicatie van de transformer (§ 2.1.3) twee jaar later, maar in de praktijk hebben onderzoekers gewenningstijd nodig gehad. Daardoor zijn veel van de technieken in §2.5 ontworpen voor de bahdanau-architectuur, met onvoorziene gevolgen van dien (§4.12).

2.1.2 2016: Sennrichs BPE

Sennrich e.a. (2016) remediëren het open woordvocabularium in de bahdanau-architectuur. Hun idee is radicaal: in plaats van woordsequenties op woordsequenties af te beelden, laten ze de encoder subwoorden lezen en de decoder subwoorden produceren, in de hoop dat die laatste een correcte zin met coherente woorden vormen wanneer aan elkaar gelijmd. Die "leap of faith" bleek een succes, en werd ook de standaard in transformers (cfr. infra). Volgens Sennrich e.a. was er voordien reeds onderzoek dat woordembeddings samenstelde op basis van subwoorden en dan op woordniveau vertaalde – een idee dat met de komst van diepe netwerken terugkomt, zie § 2.5.4 – maar dat vereist dat alle subwoorden van een woord samengevat kunnen worden in één embedding, die dan een flessenhals vormt. Zoals §1.4 beargumenteert, is het beter dat het systeem de aparte subwoorden rechtstreeks kan zien. Attention heeft er uiteraard ook baat bij om te kunnen focussen op bv. alleen de suffix van een woord i.p.v. het ganse woord.

De grootste contributie van Sennrich e.a. (2016) is echter hun subwoordtokenisatiealgoritme: *byte-pair encoding (BPE)*. De werking is in Algoritme 2.1 uiteengezet. BPE is gebaseerd op het gelijknamige compressiealgoritme van Gage (1994), alleen gebruikt het strings met letters i.p.v. rijen met bytes.

Werking BPE heeft twee fases: een *leerfase* en een *gebruiksfase*. De bedoeling van de leerfase is om V iteratief op te vullen met nieuwe subwoorden en tegelijk tokenisatieregels M te leren voor later. De gebruiksfase is wat we in Hoofdstuk 1 zagen als de uiteindelijke taak van de tokeniser, i.e. de segmentatie van een woord (hier met de geleerde V en M).

De leerfase is *ongesuperviseerd* (Eng.: *unsupervised*): ze heeft niets anders nodig dan een corpus van teksten, zonder extra grammaticale informatie of voorbeeldsegmentaties om na te streven. Het proces verloopt als volgt: het corpus wordt eerst opgesplitst in woorden (opdat we vervolgens geen subwoorden zouden leren waar spaties in zitten), en de woorden vervolgens in karakters. We initialiseren het subwoordvocabularium V met alle unieke karakters die we zo vinden (lijn 4). Vervolgens voegen we iteratief één subwoord toe aan V tot het de vooraf gekozen[11] grensgrootte τ bereikt. Het gekozen subwoord is de aaneenlijming (Eng.: *merge*) xy van *de twee bestaande subwoorden* $x, y \in V$ *die het*

[11] Sennrich e.a. kiezen voor $\tau = 30\,000$, maar zie §4.12.

Algoritme 2.1 Pseudocode BPE

```
1:  function leerBPE(𝒟, τ)
2:      𝒟 ← splitsInWoorden(𝒟)
3:      𝒟 ← [splitsInLetters(w) + [eow] | w ∈ 𝒟]
4:      V ← {ℓ ∈ w | w ∈ 𝒟}
5:      M ← []
6:      while |V| < τ do
7:          (x, y) ← arg max_{x,y∈V} aantal(𝒟, x_y)
8:          𝒟 ← vervang(𝒟, x_y, xy)
9:          V ← V ∪ {xy}
10:         M ← M + [(x, y)]
11:     return (V, M)

12: function gebruikBPE(V, M, w)
13:     w ← splitsInLetters(w) + [eow]
14:     i ← 0
15:     while i < |M| do
16:         (x, y) ← M[i]
17:         i ← i + 1
18:         if x_y ∈ w then
19:             w ← vervang(w, x_y, xy)
20:     return [index(V, t) | t ∈ w]
```

meest naast elkaar voorkomen van alle mogelijke paren (lijn 7). Vooraleer het telproces te herhalen, doen we nog twee dingen: we vervangen elk opeenvolging van de twee subwoorden x en y door het enkele subwoord xy, en we onthouden hoe xy juist ontstond[12] door (x, y) op te slaan in de *mergelijst* M.

De gebruiksfase spiegelt zichzelf aan de leerfase. Gegeven een string s zonder spaties, splitst de BPE-tokeniser s weer in karakters op, en vervolgens bootst hij de leerfase na door de merges van M in volgorde af te spelen. Daardoor vormt het iteratief grotere tokens in s, tot er geen enkel tokenpaar een merge in M heeft.
We kunnen het volledige proces visualiseren met een boom die bij de karakters begint en bij de uiteindelijke tokens eindigt, zie bv. Figuur D.7.

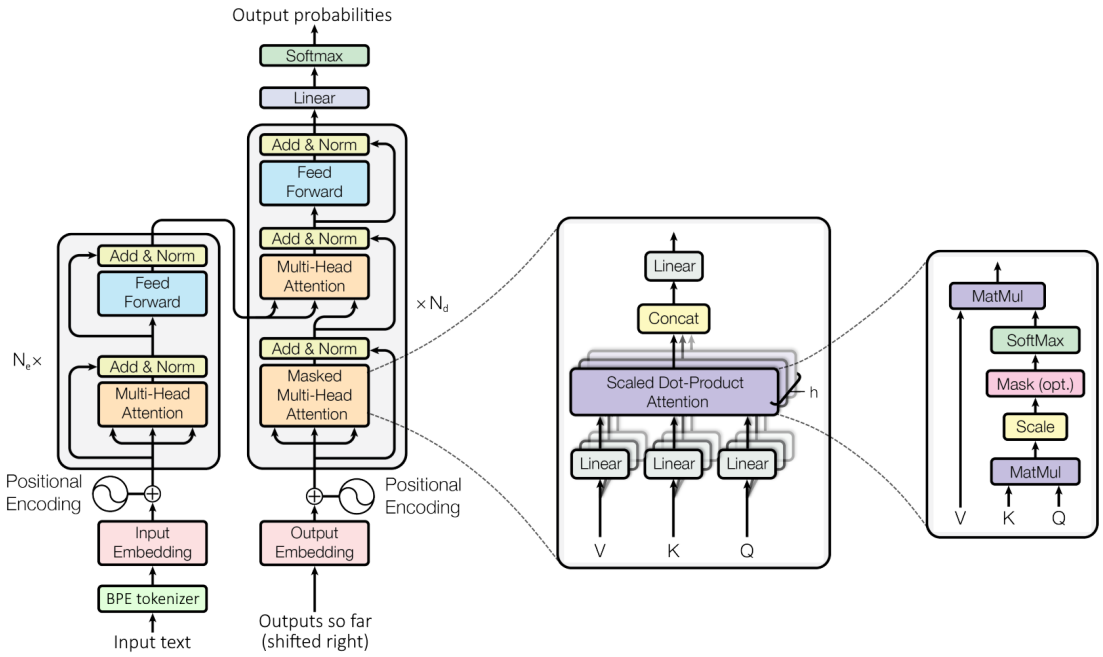
Spaties Zoals hij nu beschreven is, heeft de tokeniser geen manier om spaties tussen woorden aan te geven wanneer hij een zin segmenteert. Sennrich e.a. lossen dat op door achter elk woord eerst een speciaal *end-of-word (EOW)*-karakter te plaatsen, dat achteraf terug tot spatie kan worden omgezet. (Men zou als EOW een spatie kunnen gebruiken, maar dat is vervelend voor mensen die V willen inspecteren.) Equivalent is om te werken met een *start-of-word (SOW)*-karakter zoals "Ġ" in de BPE van RobBERT (Delobelle e.a., 2020) (zichtbaar in Figuur D.7). Nog een andere stijl is om alle subwoorden behálve het eerste een SOW te geven, zoals het voorbeeld van V op pagina 8. Een volledige bespreking houd ik voor §4.3. Let wel dat de huidige aanpak betekent dat sommige subwoorden in V als enige verschil de aanwezigheid van EOW op het einde hebben, en toch in zeer verschillende contexten voorkomen.

[12] Immers, x en y zijn zelf subwoorden die uit meerdere karakters bestaan, waardoor de merge tot xy niet inverteerbaar is. Kwam een subwoord xy = "abcd" bijvoorbeeld van x = "a" en y = "bcd", of x = "ab" en y = "cd", of x = "abc" en y = "d"?

Het recycleren van compressiealgoritmes voor tokenisatie komt verrassend veel terug, terwijl de beste manier om tekst te segmenteren helemaal niet diegene met de minste resulterende tokens is – dat zou namelijk gewoon een woordvocabularium zijn. Zo stelden Chitnis en DeNero (2015) voor om een klein deel van het eindige woordvocabularium van Bahdanau e.a. te reserveren voor een reeks betekenisloze symbolen $\sigma_1 \dots \sigma_n$, die dan gebruikt worden om codewoorden voor een grote verzameling OOV's te bouwen. Welk woord welke sequentie van codesymbolen krijgt, wordt echter bepaald met een huffmanboom, onafhankelijk van de letters van het woord, i.t.t. BPE-tokens: hun code kan de zin *"Na de examenperiode krijg ik mijn ingenieursdiploma."* bijvoorbeeld omzetten naar *"Na de* σ_1 σ_5 σ_2 *krijg ik mijn* σ_1 σ_2*."*, wat weliswaar geen ⟨UNK⟩ bevat, maar de plaatsvervangende embeddings zijn duidelijk nogal verwarrend. Ook Wu en Zhao (2018) halen inspiratie uit compressie, zie §2.5.

2.1.3 2017: Vaswani's transformer

BPE heeft kunnen meesurfen op het succes van de razend populaire[13] bahdanau-architectuur, zelfs al werd die na amper twee jaar overklast door de meest revolutionaire architectuur in de geschiedenis van NLP: de *transformer*, geïntroduceerd door Vaswani e.a. (2017). Met "revolutionair" bedoel ik dat er in de hele NLP-literatuur – en dus ook die over de tokenisatie – een heldere breuklijn loopt rond 2017: papers ervoor citeren compleet andere technologieën dan papers erna. Voordien bestond state-of-the-art-NLP uit taakspecifieke systemen gebaseerd op LSTM's en CRF's. De transformer en daaruit afgeleide taalmodellen (zie §2.2) kunnen aan veel taken worden aangepast met superieure performantie, en zijn dus overal inzetbaar.[14] Transformers liggen ook aan de basis van modellen als ChatGPT en DALL·E (cfr. Hoofdstuk 1), die voor massale en langdurige belangstelling voor NLP hebben gezorgd in oude en nieuwe media (en waarschijnlijk ook voor een grotere instroom van NLP-onderzoekers-in-spe dan ooit tevoren), waardoor NLP plots voor leken het bekendste onderzoeksdomein binnen AI is geworden.



Figuur 2.1 – De transformerarchitectuur. Combinatie van de figuren in Vaswani e.a. (2017).

[13] De paper van Bahdanau e.a. (2015) heeft een dikke 27 000 citaties volgens Google Scholar.
[14] Ter vergelijking: Vaswani e.a. (2017) zitten aan 67 000 citaties volgens Google Scholar.

Een transformer (Figuur 2.1) heeft net als Seq2Seq een encoder (linkerhelft) en een decoder (rechterhelft). Ze zijn elk een stapel van modules – een voorbeeld van *deep learning* – die net als een (Bi)LSTM op invoer met variabele lengte werken. Dat komt onder andere omdat de meeste blokjes in Figuur 2.1 apart op elke vector werken: de blauwe zijn tweelagige dense FFNN's, de grijze zijn ook dense lagen, en de gele zijn residuele connecties. De uitvoer van elke module, en dus ook de hele encoder en decoder, is een rij vectoren met dezelfde lengte als de invoer.

In de oranje blokjes interageren de vectoren met elkaar. Daar zit de kern van de transformer: *meerhoofdige, geschaalde dotproductattention* (Eng.: *multi-head scaled dot-product attention*). Om te beginnen wordt de relevantie tussen twee vectoren niet langer berekend met een leerbare functie $a(\cdot, \cdot)$ zoals Vgl. 2.1, maar met een dotproduct. Dat idee bestond al langer; Vaswani e.a. verwijzen naar de analyse van Britz e.a. (2017) waaruit blijkt dat dotproductattention degradeert voor langere vectoren t.o.v. additieve attention, wat Vaswani e.a. ertoe leidde om te delen door hun lengte (cfr infra). Attention wordt meermaals in parallel uitgevoerd in verschillende *hoofden* (Eng.: *heads*), waarbij elk hoofd één gewogen som produceert die met de rest wordt geconcateneerd.

In het midden van de decodermodules gebeurt er attention tussen de encoder en de decoder, gelijkaardig aan Bahdanau e.a. Het verschil is dat er nu niet één decodervector is, maar een ganse rij. Formeel: verzamel de vectoren van de encoder als de rijen van twee matrices $\boldsymbol{K} = \boldsymbol{V} \in \mathbb{R}^{n \times H}$, en verzamel die van de decoder in een matrix $\boldsymbol{Q} \in \mathbb{R}^{m \times H}$. In één hoofd met gewichtsmatrices $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{H \times d}$ is de attentionoperatie dan:

$$\text{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{(\boldsymbol{Q}\boldsymbol{W}_Q)(\boldsymbol{K}\boldsymbol{W}_K)^\intercal}{\sqrt{d}}\right)(\boldsymbol{V}\boldsymbol{W}_V) \qquad \in \mathbb{R}^{m \times d} \qquad (2.4)$$

... waarbij we de softmax over de rijen nemen, d.w.z. per decodervector, zoals in Vgl. 2.3. De dimensie d is gelijk aan H gedeeld door het aantal hoofden. Na concatenatie krijgen we weer m vectoren in $\mathbb{R}^H$, die als "nieuwe" $\boldsymbol{Q}$ fungeren.

De andere twee attentionblokjes doen aan *self-attention*, waarbij $\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V}$. Daarin wordt bv. elke encodervector vergeleken met *mede*-encodervectoren, inclusief zichzelf, om dan per hoofd vervangen te worden door een relevantie-gewogen som. Self-attention vervangt de bijdrage van de (Bi)LSTM's in Seq2Seq. Gezien attention echter permutatie-equivariant is (de invoer permuteren zorgt louter voor dezelfde permutatie in de uitvoer, zonder dat de vectoren veranderen), heeft het geen manier om woordvolgorde aan te geven; vandaar telt men bij de embeddings een *positionele codering* op, zoals bv. een sinusoïdale modulatie die uniek is per woordpositie.

De self-attention in de decoder is "gemaskerd": subwoorden in de uitvoerzin kunnen alleen attention voeren naar links, over de subwoorden die al eerder bestonden. Op die manier moet de decoder de embeddings van voorbije subwoorden niet herberekenen telkens er een subwoord aan de uitvoer wordt toegevoegd.

De grote lijnen van Sennrich e.a. (2016) gelden nog steeds voor Vaswani e.a. (2017): een rij BPE-subwoorden in de invoer wordt gecodeerd met een bidirectionele encoder, en het resultaat wordt autoregressief vertaald naar een andere rij BPE-subwoorden met een unidirectionele decoder. Het verschil is dat een transformer zijn attention rechtstreeks over de embeddings voert i.p.v. ze eerst te verwerken met een LSTM, en daardoor is er nu geen flessenhals meer: subwoorden zien elkaar even scherp tijdens hun interactie (een dotproduct van hun embeddings), onafhankelijk van hoeveel subwoorden ertussen liggen. Een LSTM heeft moeite met lange invoeren (bv. dankzij te granulaire tokenisatie), een transformer in theorie niet. (In de praktijk beperkt men de invoerlengte n omdat attention $O(n^2)$ ruimte en tijd vergt.)

2.2 Taal- versus vertaalmodellen

2.2.1 Algemeen

De drie papers hierboven gingen elk over een *vertaalmodel* (Eng.: *translation model, TM*): zulke systemen modelleren de kansmassa $P(y_i \mid y_{i-1} \dots y_1, x_1 \dots x_n)$ van het voorkomen van een (sub)woord y_i in een doeltaal, gegeven de voorafgaande (sub)woorden $y_{i-1} \dots y_1$ in de doeltaalzin én gegeven de volledige zin $x_1 \dots x_n$ in de brontaal. Een *taalmodel* (Eng.: *language model, LM*), daarentegen, kent meestal[15] slechts één taal, en modelleert de kansmassa van het voorkomen van een (sub)woord in die taal gegeven een context van andere woorden in die taal.

Als een taalmodel alleen context gebruikt links[16] van het subwoord, is het een *causaal LM (CLM)*. Als de context van links en rechts komt, dan is het alsof we in een doorlopende tekst een gat prikken en is het een *maskerend LM (MLM)*.

2.2.2 Transformers

Stiekem zagen we al bijna een MLM en een CLM: in een transformer is de invoer van de encoder altijd in de brontaal, en zijn de uiteindelijke vectoren het resultaat van bidirectionele attention. De "invoer" (lees: teruggekoppelde uitvoer) van de decoder is dan weer altijd in de doeltaal, en de attention is unidirectioneel.

BERT (*bidirectional encoder representations from transformers*, Devlin e.a., 2019) is een transformerencoder zonder decoder. Door een simpele FFNN-classificeerder te doen voorspellen van welk subwoord elke resulterende vector afkomstig is – en af en toe bij een masker-embedding te beginnen, zodat die vector al zijn informatie vergaart uit de context – krijgen we een MLM. Devlin e.a. kiezen voor WordPiece (cfr. §2.5) als tokeniser; RoBERTa (Liu e.a., 2019), het model dat BERT verbetert door tijdens het trainen niet altijd dezelfde subwoorden te maskeren gegeven dezelfde zin, gebruikt echter terug BPE. RobBERT (Delobelle e.a., 2020) is RoBERTa getraind op een Nederlands corpus. Er zijn ook BERT-modellen die men traint opdat de embeddings betekenisvoller zijn na combinatie (bv. uitmiddeling): SentenceBERT (Reimers en Gurevych, 2019) doet dat met een classificatietaak waarbij zinnen wel of geen parafrase zijn, en PhraseBERT (Wang e.a., 2021) veralgemeent dat naar woordgroepen. Zoals bij deep learning vaker het geval is, is het een wetenschap op zich om te achterhalen welke kennis BERT nu juist bezit en waar in het model die zich bevindt: Rogers e.a. (2021) noemen het "BERT'ologie".

GPT, GPT-2 en GPT-3 (*generative pre-trained transformer*, Radford e.a., 2019; Brown e.a., 2020) zijn allemaal transformerdecoders zonder encoder. Als CLM zijn ze voornamelijk een goed model voor NLG, gezien een CLM iteratief zijn eigen context kan blijven uitbreiden. Een MLM kan dat niet.

Een MLM is bruikbaar voor *transfer learning*: een bedrijf met veel rekenkracht traint BERT als ongesuperviseerd MLM op een grote dataset om taalkundig waardevolle vectoren te produceren, en particulieren kunnen de MLM-classificeerder en -verliesfunctie vervolgens vervangen door hun eigen tokenclassificeerder (om bv. uit een beperkte labelset een label op elke token te plakken, of bv. de zin als geheel te classificeren) die ze dan gesuperviseerd trainen op een kleine dataset en terwijl ook BERT traagjes bijstellen. Die twee fases heten *pre-training*[17] en *fine-tuning*. De tokeniser wordt (traditioneel)

[15] Een LM kan meertalig zijn (bv. mBERT van Devlin e.a. (2019), of XLM van Lample en Conneau (2019)), maar dat is nu niet pertinent.

[16] Tenzij men in die taal van rechts naar links leest, uiteraard.

[17] De P in de GPT-reeks verwijst naar het gebrek aan fine-tuning in die modellen.

vastgelegd nog voor pre-training begint, en blijft behouden tijdens fine-tuning.

Voor veel simpele taken kan een MLM (plus classificeerder) ook vervangen worden door een *ver*taalmodel. De idee is als volgt: categorische uitkomsten (bv. klassen) worden bij het trainen zelf als volgnummer voorgesteld, maar als we de accuraatheid overheen de uitkomsten rapporteren om door mensen gelezen te worden, dan gebruiken we geen volgnummers maar klassenamen. Wat als we de decoder van een vertaalmodel geen tweede taal laten spreken, en in plaats daarvan de "taal" die bestaat uit de klassenamen? Dat is exact het uitvoerformaat waarop T5 (*text-to-text transfer transformer*, Raffel e.a., 2020), een gewone transformer (ook met BPE), getraind is. Sterker nog: in plaats van één transformer (of op z'n minst de decoder) te trainen per taak, geeft men in de invoertekst telkens een speciaal token mee dat de huidige taak aangeeft. T5 toont dus niet alleen dat een vertaalmodel ingezet kan worden waar één taalmodel gebruikt werd, maar meerdere tegelijk, en allemaal gewoon door patronen in de dataset te steken.

2.2.3 BLEU

Men beoordeelt en vergelijkt TM's altijd op basis van de BLEU-metriek van Papineni e.a. (2002), die aantonen dat BLEU sterk correleert met de evaluatie van menselijke jury's. Echter, het is niet alleen traditie om de vertalingen zelf met BLEU te beoordelen (*intrinsieke evaluatie*), maar ook de aparte onderdelen van het vertaalmodel, die in principe niet per se voor vertaling dienen (*extrinsieke evaluatie*). Zo worden alle tokenisers in §2.5 gestaafd met BLEU in hun papers, en niet met een metriek die schat hoe morfologisch correct de segmentaties zijn.

De redenering is dat "hogere BLEU ⇒ betere onderdelen voor MT", wat klopt, maar dat wordt dan doorgetrokken naar "betere onderdelen" tout court, bv. voor NLU in LM's, waar BLEU geen betrekking op heeft. Dat kan op zijn beurt leiden tot het geloof dat "betere onderdelen ⇒ hogere BLEU", waardoor tokenisers die waardevoller zijn in de praktijk niet opvallen en men er niet over publiceert.

BLEU meet de *precision* van woorden[18] in de voorspelde vertaling, d.w.z. het percentage van woorden in de voorgestelde vertaling dat ook in de referentievertaling aanwezig is. Dat is problematisch. Zo merken Sennrich e.a. (2016) op dat BLEU zeer weinig informatie geeft over de mate waarin cruciale woorden beter dan voordien worden vertaald. Vandaar meten ze hun verbetering t.o.v. Bahdanau e.a. (2015) met nog een aparte F_1-score specifiek voor zeldzame woorden.[19] Dat is niet verwonderlijk, gezien Sennrich e.a. met Engels→Duits-vertalingen werken, terwijl de correlaties in Papineni e.a. (2002) alleen voor Chinees→Engels gelden (en in het verdere onderzoek dat ze vermelden, beschouwen ze Romaanse i.p.v. Germaanse talen). In dezelfde lijn vermelden Huck e.a. (2017) dat ze tot winnaar van een vertaalcompetitie met menselijke jury werden verkozen, terwijl hun BLEU-score niet de hoogste was. Anderzijds merken Sennrich e.a. dat hun model tot 1 eenheid[20] BLEU varieert door louter willekeur in het trainen en decoderen.

Zelfs áls de BLEU-metriek een goede extrinsieke maatstaf was, dan nog is een vergelijking tussen twee papers onbetrouwbaar, vermits de berekening van BLEU meerdere vrijheidsgraden kent. Dat komt omdat Papineni e.a. (2002) reeds meerdere parameters in BLEU

[18] Meestal ook woordparen (2-grams), woordtrio's (3-grams) en woordkwartetten (4-grams). De maximumgrootte is een instelbare parameter, zie verder.

[19] In praktische vertaalapplicaties houdt dat ook meer steek, zeker aangezien de zeldzame woorden in een zin juist de kern van de betekenis bevatten en dus cruciaal zijn voor de kwaliteit.

[20] Strikt genomen ligt BLEU tussen 0 en 1. Tegenwoordig citeert men echter steeds waarden rond de 30, en dat zijn dan percentages zonder %-teken. "1 BLEU" is vandaag dus 1% precision – hetgeen men als significante toename ziet – terwijl dat vroeger 0.01 BLEU was (zie bv. Koehn en Knight, 2003).

inbouwden, en Post (2018) toont bovendien dat zowel de keuze van pre-tokeniser als de aanwezigheid van ⟨UNK⟩'s voor grote variaties in BLEU zorgen.

2.3 Embeddings

Het loont om kort toe te lichten wat met "embeddings" bedoeld wordt in al het bovenstaande. De betekenis van een ((sub)woord)type wordt met een lijst van d reële getallen voorgesteld die gradaties van onbepaalde eigenschappen voorstellen: de lijst $(4.20; 0; -69; \pi)$ heeft een "hoedanigheid" van 4.20 in de eerste "eigenschap", geen in de tweede etc... Vaak lenen we vervolgens de bewerkingen van vectorruimtes met $\mathbb{L}^2$-norm om dergelijke reële lijsten te laten interageren, maar het is naïef om te denken dat er enige link is tussen taal en de $\mathbb{L}^2$-ruimte, zie §2.4.2. Een dergelijke inbedding als vectoren heet een embedding.

Om die embeddings op te vullen zijn er *spaarse* en *dense* methodes. De meest spaarse embeddingmethode heet *one-hot*, waarbij elk type in V zijn eigen dimensie krijgt en in elke dimensie 0 heeft buiten in die ene; daar heeft het waarde 1. Daarvan afgeleid is de minder spaarse *TF-IDF*-embedding waarbij men voor elk document in een dataset van d documenten de one-hot-embeddings van de types in dat document optelt, en de resultaten als rijen van een $d \times |V|$-matrix verzamelt. De d-dimensionale kolommen geven dan per type aan in welke documenten ze voorkomen.

Hoe minder spaars embeddings zijn, hoe meer ze zich verlenen aan elementsgewijze vergelijkingen om patronen te herkennen. Twee TF-IDF-embeddings die in gelijkaardige documenten voorkomen, hebben een gelijkaardige embedding en hebben (volgens de DH) waarschijnlijk een gelijkaardige betekenis. Een manier om dense embeddings af te dwingen buiten de setting van documenten is om d veel kleiner dan $|V|$ te maken, opdat de types volgens het *duivenhokprincipe* niet allemaal hun eigen dimensie kunnen claimen en er dus correlaties ontstaan.

Statische embeddings zoals de twee in de *word2vec*-paper (Mikolov e.a., 2013a) zijn type-centrisch: alle tokens van hetzelfde type worden voorgesteld met dezelfde embedding. Mikolov e.a. trainen een heel klein neuraal LM met een zeer beperkte context, en nemen de interne gewichten van dat LM als embeddings. Pennington e.a. (2014) nemen met hun *GloVe*-embeddings een meer theoretische aanpak en herleiden het zoeken naar statische embeddings tot een kleinstekwadratenprobleem: als de woorden w_i en w_j in het corpus $C_{i,j}$ keer in elkaars buurt voorkomen, dan is het doel om hun embeddings zo af te stellen dat het dotproduct $\ln(C_{i,j})$ benadert.

Gecontextualiseerde embeddings zoals de uitvoer van BERT (Devlin e.a., 2019) zijn daarentegen token-centrisch: afhankelijk van de context waarin een token zich bevindt, krijgt het een andere embedding – bv. onder invloed van attention. Vaak noemt men als voordeel van contextualisatie dat het *homografie* en *polysemie* kan modelleren, i.e. dat één type meerdere betekenissen kan hebben.[21] Contextualisatie zorgt echter voor veel rijkere informatie, bv. syntactische verbanden. Indien niet, zouden NMT-decoders niet zo goed kunnen werken. Ook durft men beweren (Rogers e.a., 2021) dat de contextuele embeddings van de tokens van één type zich clusteren rond B vectoren indien dat type B aparte betekenissen heeft, maar Ethayarajh (2019) toont dat het juist de minst ambigue types zijn (*"is"*, *"en"*, *"de"* ...) wiens tokens de meeste variatie kennen.

[21] Men verwart homografie en polysemie wel eens. Homografie is per ongeluk: zo zit er *bloem* in brood en groeit een *bloem* in de wei,[22] maar bloem komt niet van bloemen. Polysemie is met opzet: een kleuter kan *oortjes* in haar *oortjes* hebben om naar muziek te luisteren.

[22] Toevallig zijn de Engelse vertalingen, *flour* en *flower*, homofonen.

Is de studie van statische embeddings achterhaald nu we contextuele embeddings hebben? Zeker niet, want subwoordmodellen beginnen altijd met een embedding-LUT die voor alle tokens van hetzelfde type dezelfde embedding geeft, en in sommige contextualiserende modellen zitten er zelfs meer parameters in de embedding-LUT dan de rest van het model (Chung e.a., 2021).
Een recente statische methode is *joint spherical embedding (JoSE)* van Meng e.a. (2019): de clou van hun verhaal is dat embeddings meestal ontworpen zijn voor gebruik in een LM of in MT, en niet om rechtstreeks vergeleken te worden. In clusteringanalyses, bv. om overkoepelende topics te ontdekken (Sia e.a., 2020; Thompson en Mimno, 2020), is dat juist exact wat men wil doen. Vergelijken doet men met cosinussimilariteit, waarbij een normalisatie plaatsvindt:

$$\cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\|\vec{y}\|} = \frac{\vec{x}}{\|\vec{x}\|}\cdot\frac{\vec{y}}{\|\vec{y}\|} = \frac{\vec{x}}{\sqrt{\vec{x}\cdot\vec{x}}}\cdot\frac{\vec{y}}{\sqrt{\vec{y}\cdot\vec{y}}} \tag{2.5}$$

Gewoonlijk (bv. in word2vec en GloVe) worden embeddings geleerd zonder de beperking dat ze genormaliseerd moeten zijn, dus mogelijks zijn ze nadien niet meer optimaal. Het zou echter voor lastige gradiënten zorgen moesten we Vgl. 2.5 rechtstreeks in een verliesfunctie opnemen, dus verzinnen Meng e.a. een nieuw algoritme waarin de normalisatie ingebouwd is. De resultaten zijn veel beter dan pure BERT-embeddings op allerlei similariteitstaken na minder dan een uur training.

2.4 Grote woorden voorstellen

De ideaal die subwoordtokenisatie nastreeft, is om eender welk product van woordformatie (samenstelling en afleiding) te voorzien van een semantische voorstelling, nl. de rij van de embeddings van de relevante subwoorden. We kunnen ons afvragen hoe dergelijke rijen zich gedragen t.o.v. de embeddingruimte: stel dat het vocabularium de subwoordtypes {*schoon*, *-heid*, *schoonheid*} bevat, wat voor verband willen we dan hebben tussen $[\vec{v}_{\text{schoon}}, \vec{v}_{\text{-heid}}]$ en $\vec{v}_{\text{schoonheid}}$?

De expressiviteit van de embeddingruimte wordt beperkt door het aantal embeddingsdimensies, dus er moeten concepten zijn die niet in één embedding vatbaar zijn. Intuïtief is de hoogst mogelijke expressiviteit alleen te bereiken als de subwoorden in de rij elk met een zo helder mogelijk concept overeenkomen: het loont dus om de literatuur over woordgebaseerde tokenisatie – d.w.z. dat de tokenisatie van een groot woord resulteert in kleinere maar intacte woorden – te verkennen. Bovendien kunnen we zo'n woordtokeniser inbouwen aan het begin van een subwoordtokeniser om het behoud van de belangrijkste morfeemgrenzen te garanderen (Huck e.a., 2017).

2.4.1 Strings splitsen

Koehn en Knight (2003) introduceren de *frequentiemethode* om samenstellingen te splitsen. Die bestaat erin om voor een woordtype s elk van de $2^{|s|-1}$ mogelijke segmentaties op te lijsten[23] en ze te rangschikken volgens het geometrisch gemiddelde van de frequenties van hun onderdelen. De score van een segmentatie $s = s_1 \dots s_m$ is m.a.w.

$$\text{score}(s_1,\dots,s_m) = \sqrt[m]{\prod_{i=1}^{m} C_{\mathcal{D}}(s_i)} = \exp\left(\frac{1}{m}\sum_{i=1}^{m}\ln C_{\mathcal{D}}(s_i)\right) \tag{2.6}$$

[23] Na elke letter, buiten de laatste, kan één of geen splitsing komen. Koehn en Knight schrijven dat dit een goede toepassing van "dynamic programming" had kunnen zijn, maar dat ze niet om efficiëntie geven. Bij mijn weten heeft niemand hun methode uiteindelijk efficiënter geïmplementeerd.

... waarin $C_{\mathcal{D}}(s_i)$ het aantal alleenstaande tokens s_i in corpus $\mathcal{D}$ is. Merk op dat een gebrek aan splitsing ($m = 1$) eveneens wordt gerangschikt, en deze heuristiek dus zowel samenstellingen detecteert als een splitsing suggereert. Vandaar introduceren de auteurs ook nieuwe definities van *precision* en *recall* die de twee taken combineren:

$$\begin{aligned} P &= \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\#(\text{correct gesplitst} \mid \text{te splitsen})}{\text{TP} + \#(\text{fout gesplitst} \mid \text{te splitsen}) + \#(\text{gesplitst} \mid \text{niet te splitsen})} \\ R &= \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\#(\text{correct gesplitst} \mid \text{te splitsen})}{\text{TP} + \#(\text{fout gesplitst} \mid \text{te splitsen}) + \#(\text{niet gesplitst} \mid \text{te splitsen})} \end{aligned} \tag{2.7}$$

Zoals gewoonlijk zijn precision en recall een betere maatstaf dan *accuracy* vanwege een onbalans in klassen; Koehn en Knight geven bv. toe dat 94% van hun manueel gesplitste dataset uit ongesplitste woorden bestaat, en een systeem dat nooit splitst dus reeds "94% accuraat" is.

Ze tonen ook dat de frequentiemethode niet 100% betrouwbaar is. Enerzijds worden veelvoorkomende samenstellingen (bv. *actieplan*, frequenter dan *actie* en *plan*) niet gesplitst, en anderzijds hallucineert een segmentatie soms over hoogfrequente deelwoorden (bv. "*en*" en "*de*" zoals in *volgende*, of *vrij* en *dag* zoals in *vrijdag*) die dan foutief aanzetten tot splitsen. Een eerste verbetering die ze voorstellen is om een interfix-*s* te tolereren door een *s* links van een splitsing al dan niet te negeren. Een tweede verbetering is om alleen deelwoorden met een *woordsoort* (Eng.: *part of speech, PoS*) uit een klein groepje toe te laten (± substantief, adjectief of werkwoord, en geen voorzetsel noch lidwoord). Een laatste verbetering gaat uit van een parallel corpus waarin de te splitsen samenstelling reeds aanwezig is, maar dat is mijns inziens valsspelerij gezien we het open vocabularium dan modelleren door het gewoon deels op te lijsten.

Stymne (2008) stelt extra PoS-beperkingen voor, bv. dat het meest rechtse deelwoord dezelfde PoS moet hebben als de samenstelling toegewezen krijgt o.b.v. haar context. Ze breidt de interfixregels ook uit door letterverwijderingen links van een splitsing toe te laten (zoals bv. het Duitse *Kirche* in *Kirchhof*).

Fritzinger en Fraser (2010) merken een paradox in de literatuur: systemen gebaseerd op de frequentiemethode lijken beter te presteren bij extrinsieke evaluatie met BLEU (voor zover mogelijk, cfr. § 2.2.3), maar slechter bij intrinsieke evaluatie met Vgl. 2.7. Hun idee is om de twee samen te brengen: in plaats van álle mogelijke segmentaties te rangschikken met Vgl. 2.6, beschouwen ze alleen dié splitsingen die een (bestaande) morfologische analysator suggereert. Dat kunnen er meerdere zijn: de analysator is in dit geval gebaseerd op een *finite-state machine (FSM)* die bijvoorbeeld het woord *rivierbeddingsdoorsneden* van links naar rechts opbreekt als *rivier|bed|ing|doorsnede* of[24] als o.a. *rivier|bed|ing|door|snijden*. Merk op dat de uitvoer de interfix verwaarloost en met lemmata werkt; de auteurs zetten achter de analysator een postprocessor die lemmata terug naar de respectieve substrings omzet (*rivier|bed|ing|door|sneden*), vervolgens alle onzelfstandige woorden terug samenvoegt (*rivier|bedding|door|sneden*), en tot slot alle mogelijke combinaties van samenvoegingen oplijst (*rivier|bedding|door|sneden*, *rivierbedding|door|sneden*, *rivier|bedding|doorsneden*, *rivierbedding|doorsneden* ...).

Sommige woorden in hun manuele dataset die hun systeem correct splitst, splitste dat van Stymne fout: van het Duitse voorbeeld *entbrennen* ("ontspringen") maakte het bv. *Ent(e)+brennen* ("eend-branden"),[25] wat niettemin de PoS-beperkingen van zowel Koehn en Knight als Stymne gehoorzaamt. Soms zijn de PoS-regels wel genoeg: een Duitse analysator weet dat *traumatisch* een adjectief is (want het heeft geen hoofdletter)

[24] Aangezien "sneden" het meervoud is van "snede", maar ook de verleden tijd van "snijden".
[25] Vandaar de titel van hun paper: *"How to Avoid Burning Ducks"*.

en dus niet gelijk kan zijn aan het esoterische substantief *Trauma+Tisch* ("traumatafel"). Fritzinger en Fraser merken ook op dat de frequentiemethode zonder analysator niet bestand is tegen een corpus met *syllabificatie*, het proces in geprinte tekst dat een woord op de grens van twee lettergrepen doorsnijdt omdat het anders te lang is om op dezelfde regel te passen. Bij het digitaliseren kunnen die twee delen per ongeluk als twee aparte woorden tellen; in een taal met regelmatige uitgangen is het afgesplitste deel vaker hetzelfde, dus tellen uitgangen plots mee als hoogfrequente woorden.

Verder doen ze een gedetailleerde analyse van de oorzaken van foute splitsingen (zie hun tabel 5). De meeste fouten komen nog steeds door hallucinaties: *nadeel* had in theorie van *na+deel* kunnen komen – naar analogie met *nabespreking* – zonder de PoS-regels te schenden, en het wordt gekozen doordat "*na*" zo hoogfrequent is. Ook verrassend: in slechts 3% van alle fouten (22 van de 741) bevond de juiste splitsing zich niet tussen de gerangschikte segmentaties, wat dan komt door foute interpretatie van de analysator. In het bijzonder gebeurde het *slechts 3 keer* dat hij een gelexicaliseerd woord (*muggenziften*, *kunstgras* ...) niet kende en voorstelde om het te splitsen (en/of een foute PoS voorstelde). Waarom zo weinig? Omdat het gelexicaliseerde deel van de taal eindig is (afgezien van neologismen die men gaandeweg uitvindt): als er namelijk oneindig veel woorden waren waarvan de betekenis niet af te leiden was uit hun morfologie, dan moesten woordenboeken of medemensen oneindig veel woorden definiëren. Dat kan niet, dus is het haalbaar om in zo'n analysator alle gelexicaliseerde woorden op te slaan.

Macken en Tezcan (2018) bouwden recent (i.f.v. een systeem dat naar vakterminologie zoekt) een splitser voor het Nederlands o.b.v. de ideeën van Koehn en Knight en Stymne. Ze gebruiken een *prefixboom* (Eng.: *trie*) om alle mogelijke kop- en staartwoorden[26] van een samenstelling op te slaan, en gaan dus uit van binaire splits, al experimenteren ze met recursie.[27] De bladeren stellen (woord,PoS)-tuples voor, dus heeft een woord als *groen* of *werk* twee bladeren met twee frequenties – t.o.v. de vorige systemen, die altijd de meerderheids-PoS gebruikten.

Ze passen nog veel meer heuristieken toe. De bomen worden meermaals gefilterd: ze houden enkel woorden met frequentie $\geq$20, enkel woorden met minstens drie letters en één klinker, geen bekende suffices (-heid, -ing ...) of voegwoorden (per, hoe, als ...), geen restanten van syllabificatie (manueel gefilterd), en geen bijwoorden of voorzetsels als staarten. Gevonden splitsingen worden ook gefilterd: enkel bepaalde combinaties van kop- en staart-PoS, enkel zonder interfix-*s* als er bepaalde letters rondstaan en de kop geen adjectief, bijwoord of voorzetsel is, en, als de kop op letter *ℓ* eindigt, geen drieletterige staart -*ℓ*en (zoals in *boodschappen* versus *boodschap-pen*). Toch zijn er voorbeelden die hun heuristieken omzeilen: *mopperen* wordt *mop-peren*, en *receptoren* wordt *recept-oren*. Ze vonden ook dat de verwarring tussen de interfix-*s* en meervouds-*s* tot 50% van de resterende fouten kon verklaren.

Gezien hun systeem enkel met absolute woordaantallen werkt, ondersteunt het gemakkelijk *domeinadaptatie*. De bomen komen van een groot corpus (Wikipedia), en zijn dus niet gevoelig genoeg voor specifieke vakgebieden; relevante corpora zijn echter klein, dus hun woordaantallen bij die van het grote corpus voegen heeft weinig invloed. Vandaar berekenen Macken en Tezcan *relatieve* frequenties in het kleinere corpus, om die vervolgens te herwegen met een groter gewicht. In §4.9 zal domeinadaptatie – een vorm van transfer learning – minder evident blijken voor BPE.

[26] Hun terminologie is enigszins tegenstrijdig met andere literatuur die de kern van de samenstelling (het achterste woord) "head" noemt, terwijl de kop hier het *eerste* deel is.

[27] Het is onduidelijk of ze zowel voor kop als staart recursief splitsen. Samenstellingen zijn meestal genesteld van rechts naar links (bv. ␣␣␣*baar*␣*moeder*␣*hals*␣*kanker*), dus de kop verder splitsen is nuttig. Het is echter onduidelijk of zo'n recursief gesplitste kop zelf reeds gekend moet zijn.

Huck e.a. (2017) maken de brug van de bovenstaande literatuur uit de wereld van woordgebaseerde SMT naar subwoord-NMT. Zij splitsen suffices (en al dan niet prefices) af o.b.v. een manueel samengestelde lijst, vooraleer samenstellingen te splitsen en tot slot BPE toe te passen op de resterende strings. Meer daarover in § 2.5.3. In de follow-up van Zhou (2018) blijkt het soms voordelig om die splitsingen terug samen te voegen. Let wel op: Fritzinger en Fraser (2010) raden ook aan om minder splitsingspunten te zetten (door de grootste toegelaten m in Vgl. 2.6 verlagen) omdat dat voor een hogere precision zorgt in Vgl. 2.7, maar zij moeten zich geen zorgen maken over fouten die BPE achteraf kan maken. Waar Fritzinger en Fraser tevreden zouden zijn met de splitsing *rodewijn|avond*, moeten Huck e.a. oppassen dat BPE daar bv. niet *rod|ew|ijn|av|ond* van maakt. Het is beter om zo veel mogelijk grenzen af te bakenen (bv. *rode|wijn|avond*) opdat BPE de kans niet krijgt om twee deelwoorden samen te voegen (zie ook §4.16).

Voor zover ik weet bestaat er geen literatuur die PoS-informatie gebruikt om BPE te verrijken, terwijl we hierboven toch het nut zagen. Nog enkele artificiële voorbeelden:

> Men spreekt best niet *kinderlijk*$_{\text{bijw}}$ over een *kinderlijk*$_{\text{subst}}$.
> De atleten *verspringen*$_{\text{ww}}$ in het klassement van het *verspringen*$_{\text{subst}}$.
> Het kan *voorkomen*$_{\text{ww}}$ dat men moet *voorkomen*$_{\text{ww}}$ vanwege het *voorkomen*$_{\text{subst}}$ van een gerechtelijk onderzoek.

waarin alleen de homografen met een groen PoS-label compositioneel te splitsen zijn.

Merk tot slot op dat geen van de bovenstaande methodes robuust is tegen spelfouten. Een redacteur van een boek, paper, wettekst, krantenartikel ... zou iets als "*schoonhijt*" nooit doorlaten, maar dat kan wel gebeuren in chatgesprekken of berichten op sociale media. Menselijke interpretatie is in de meeste gevallen robuust tegen imperfecte taal, dus zou een ideaal LM dat ook moeten zijn.

2.4.2 Embeddings combineren

We weten dat mensen oneindig veel concepten kunnen beschrijven en mentaal kunnen voorstellen. Computationeel willen we dezelfde macht hebben, en al die betekenissen kunnen voorstellen en vergelijken in een vectorruimte. Complexe ideeën die men als één woord schrijft, kunnen we – en moeten we, door het eindige vocabularium – opbreken in meerdere kleine ideeën, zoals hierboven. Er zijn evengoed talen waar dat standaard zo is (het Engels), en algemener zijn er veel ideeën die men alleen met een zin kan beschrijven. We zagen reeds (sub)woordembeddings (al dan niet gecontextualiseerd), maar die willen we dus tot grotere ideeën combineren. Indien we dat niet kunnen, dan is dat een zware handicap van tokenisatie en/of vectormodellen.[28]

Ik alludeerde er eerder al op dat men her en der *wishful thinking* durft uiten omtrent het componeren van embeddings. Het komt immers veel voor dat men een LM loslaat op subwoorden, en zich dan afvraagt hoe men het resultaat op *woord*niveau kan interpreteren. De mythe gaat dat men als embedding van een woord "gewoon" het gemiddelde van de embeddings van zijn subwoorden kan gebruiken (zie bv. Thompson en Mimno, 2020). Men hoopt m.a.w. dat lexicale compositie hetzelfde is als optelling in $\mathbb{L}^2$. Dat is dubieus om meerdere redenen. Een wiskundig tegenargument besprak ik reeds in de context van JoSE, nl. dat men embeddings met cosinussimilariteit vergelijkt (zo ook in Thompson en Mimno, 2020) en dus in het sferische domein, terwijl het gemiddelde van twee vectoren louter op hun bissectrice ligt wanneer ze even lang zijn. Anders liggen $\vec{x}+\vec{y}$ en $\vec{x}/\|\vec{x}\|+\vec{y}/\|\vec{y}\|$ niet in elkaars verlengde.

[28] Beperkt zijn ze sowieso, cfr. de problematiek van de "information bottleneck".

Figuur 2.2 toont een geometrisch voorbeeld waarin de "semantische algebra" niet intuïtief werkt: stel dat $\vec{x}$ de embedding van *blij* is. Wat is de similariteit met *boos*? Is ze -1, d.w.z. dat ze elkaars tegengestelde zijn en *boos* dus $-\vec{x}$ als embedding heeft, of 0, d.w.z. dat ze orthogonaal staan zoals $\vec{y}$ in de figuur? Semantisch lijkt $-\vec{x}$ beter, maar coördinatengewijs is $\vec{y}$ de juiste embedding van *boos*, orthogonaal op *blij*. Bertels (2022) bemerkt zo ook dat een *patiënt* en een *dokter* tegengestelde rollen hebben, maar in veel gelijkaardige contexten voorkomen en dus juist een hoge similariteit hebben.

Een semantisch argument tegen embeddingalgebra is dat er meer dan één compositionele relatie bestaat: een afleiding gelijkt meer op haar prefix dan haar suffix. Een *jager-verzamelaar* is het gemiddelde tussen een jager en een verzamelaar, maar een *strandhuis* deelt geen van zijn kenmerken met een strand (bv. in grootte, materiaal ...). Shwartz en Waterson (2018) bemerken dat de *olie* in *olijfolie* en *babyolie* zich (gelukkig) helemaal anders verhoudt tot de andere deelwoorden. Over dat soort relaties zo meteen meer.

De belangrijkste opdeling die ik in de literatuur hierover zie, is het *leren samenstellen* van embeddings enerzijds, en het *leren van samenstelbare* embeddings anderzijds.[29]

2.4.2.1 Compositiefuncties leren

Mikolov e.a. (2013b) menen dat word2vec-embeddings (Mikolov e.a., 2013a) een zekere semantische algebra vertonen m.b.t. optellen en aftrekken:

$$\vec{v}_{\text{koning}} - \vec{v}_{\text{man}} + \vec{v}_{\text{vrouw}} \approx \vec{v}_{\text{koningin}}$$
$$\vec{v}_{\text{Berlijn}} - \vec{v}_{\text{Duitsland}} + \vec{v}_{\text{Frankrijk}} \approx \vec{v}_{\text{Parijs}}$$
$$\vec{v}_{\text{Brussels Airlines}} - \vec{v}_{\text{België}} + \vec{v}_{\text{Griekenland}} \approx \vec{v}_{\text{Aegean Airlines}}$$

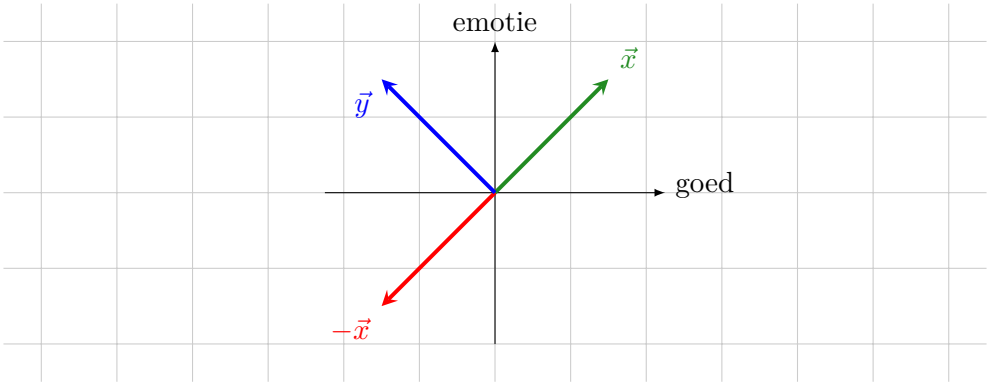
Dat laatste is echter niet veralgemeenbaar, gezien dat vereist dat elke meerwoordige uitdrukking (Eng.: *multiword expression, MWE*) in het vocabularium zit en genoeg voorkomt in het trainingcorpus om een embedding te leren. Salesky e.a. (2018), die een bahdanau-architectuur trainen waarin de BPE-tokeniser slechts gradueel meer en meer merges mag doen (zie §2.5.2.2), experimenteren met verschillende manieren om de embedding van een nieuw subwoord te initialiseren o.b.v. de embeddings van de twee gemergde subwoorden tot dan toe: ze ondervinden dat een gemiddelde zich quasi gelijk met een willekeurige initialisatie gedraagt, en dat het beter werkt om een zeer expressieve, trainbare functie te gebruiken, met name een drielagige *auto-encoder (AE)*.[30] Verder tonen Ács e.a. (2021) voor verschillende woordclassificatietaken (toewijzen van PoS-labels, morfologische labels, of herkennen van eigennamen) dat het gemiddelde van de subwoorden binnen een woord telkens slechter presteert dan expressievere compositiefuncties zoals een BiLSTM of een gewogen gemiddelde met neuraal bepaalde gewichten.[31]

Yazdani e.a. (2015) leren een compositiefunctie voor het geval van een tweeledige samenstelling (Eng.: *noun-noun compound, NNC*) met een algemene lineaire regressie, $\vec{y} = \boldsymbol{W}\vec{x} + \vec{\varepsilon}$. Dat betekent dat ze in de trainingdata van die regressie zowel de embeddings van de deelwoorden (geconcateneerd tot $\vec{x}$) als van de samenstelling ($\vec{y}$) nodig

[29] Opgelet: dat de embeddings in het eerste geval niet geleerd worden i.f.v. het samenstellen, betekent niet dat het altijd over statische embeddings gaat. Ács e.a. (2021) bestuderen bv. contextuele BERT-embeddings zonder finetuning.

[30] Een AE is een diep neuraal netwerk dat als uitvoer simpelweg zijn invoer moet reproduceren. De middelste laag is echter van een veel lagere dimensie dan de buitenste lagen, dus leert het netwerk een soort verliesloze compressie- en decompressiefunctie.

[31] Dat laatste noemen Ács e.a. "attention" verwijzend naar Vgl. 2.3, maar in hun geval heeft Vgl. 2.1 slechts één vector als argument. Er gebeurt dus geen attention tussen subwoorden. Dat is onduidelijk in de paper, maar J. Ács heeft die werking in persoonlijke correspondentie bevestigd.



Figuur 2.2 – Semantische algebra, of niet?

hebben: ze halen zo'n 400 000 NNC's uit een Wikipedia-corpus en voegen de 70 000 frequentste toe aan hun word2vec-vocabularium opdat die "superwoordtypes" ingebed worden als $\vec{y}$. Voor de rest van de data, die wel een $\vec{x}$ maar geen $\vec{y}$ kennen, maken ze hun regressieobjectief semi-gesuperviseerd met een *autoreconstructie*term die wenst dat de hypothetische compositie $\boldsymbol{W}\vec{x}$ ook terug naar $\vec{x}$ kan worden omgezet. Het overeenkomstige model is $\vec{x} = \boldsymbol{W}^\intercal(\boldsymbol{W}\vec{x}) + \vec{\varepsilon}$. Als laatste maken ze hun objectief robuust tegen uitschieters door de regressie meermaals uit te voeren en elke iteratie dié NNC's te negeren die de vorige iteratie een te grote afwijking $\|\vec{\varepsilon}\|$ t.o.v. hun $\vec{y}$ hadden; hun methode detecteert dus NNC's die waarschijnlijk non-compositioneel zijn. Eerder werd dat reeds door Weller e.a. (2014) gedaan via cosinussimilariteit tussen de embedding[32] van een NNC en van de deelwoorden die de splitser van Fritzinger en Fraser (2010) suggereert. Shwartz en Dagan (2019) voorzien een evaluatieframework waarin (non-)compositionaliteitsdetectie in zes gerelateerde taken is opgedeeld.

Dao e.a. (2021) trainen andere compositiefuncties op dezelfde soort $(\vec{x}, \vec{y})$-paren als Yazdani e.a. (2015), alleen willen ze zo'n functie vervolgens gebruiken als bevroren component gevolgd door classificatie van de *NNC-relatie* (bv. "gemaakt van" voor *olijfolie* en "bedoeld voor" voor *babyolie*). Hun resultaten zijn zwak, zelfs als ze rechtstreeks $\vec{y}$ classificeren i.p.v. $f(\vec{x})$, wat erop wijst dat word2vec-embeddings zich niet gedragen zoals we intuïtief willen. Het Nederlandse systeem van Verhoeven e.a. (2012) is analoog, alleen gebruiken zij een soort bag-of-words als embeddings en *principalecomponentenanalyse (PCA)* als bevroren functie. Met een 1-vs-1-SVM halen ze amper 50% accuraatheid.

Dima en Hinrichs (2015) vervangen dat geheel door GloVe-embeddings gevolgd door een tweelagig NN; i.t.t. de aanpak van Verhoeven e.a. of Dao e.a. (eerst een algemene compositiefunctie trainen, die bevriezen, en de uitvoer dan leren classificeren) leren zij hun compositie i.f.v. de eindtaak. Ze maken de middelste laag van het NN even groot als één embedding, opdat de activatie kan dienen als embedding van de NNC zelf. Die activatie blijkt eigenlijk vooral informatie over de NNC-relatie te bevatten, daar ze gelijkaardig blijkt te zijn voor NNC's met dezelfde relaties – maar niet per se dezelfde woorden. Dat is goed voor de classificatie, maar volgens Shwartz en Waterson (2018) leert de eerste NN-laag om telkens één van de twee woorden in $\vec{x}$ te herkennen en het andere te negeren, wat op overfitting wijst. Vandaar bouwen ze een extra invoervector die samenvat hoe de woorden zich tot elkaar verhouden als zinsdelen in alle zinnen waarin ze voorkomen maar geen samenstelling zijn.

[32] De auteurs gebruiken in feite een soort GloVe avant-la-lettre als embeddings.

2.4.2.2 Compositie-embeddings leren

De embeddings in §2.3 liggen in een ruimte waarin een gemiddelde nemen betekenisloos is. Echter, als men het objectief van hun leerproces laat afhangen van dat gemiddelde, dan zullen de embeddings zich tijdens het trainen noodzakelijkerwijs zo moeten schikken dat hun gemiddelde wel betekenis krijgt. Dat gaat dan ten koste van de individuele embeddings.

Dima en Hinrichs (2015), wiens objectief het classificeren van de NNC-relatie is, verkennen bv. wat er gebeurt indien ze hun initiële GloVe-embeddings mee fine-tunen met het NN erachter. Volgens hen is dat voordelig indien de initiële embeddings op een klein corpus zijn getraind, en/of indien ze een kleine lengte hebben.

Reimers en Gurevych (2019) fine-tunen BERT met het objectief dat twee zinnen met gelijkende betekenis ook een gelijkende gemiddelde embedding hebben, en omgekeerd voor willekeurige zinnen. Wang e.a. (2021) veralgemenen dat naar woordgroepen en hun parafrases. Ze zijn extra voorzichtig dat een woordgroep en haar parafrases geen enkel sleutelwoord gemeen hebben, opdat het onmogelijk is om te vergelijken o.b.v. woordoverlap (waar Reimers en Gurevych, 2019 niet op letten). De compositie moet echter niet per se ná de transformer gebeuren: Casas e.a. (2020) passen de architectuur van Figuur 2.1 aan door, na een aantal encodermodules, de subwoorden te combineren tot woorden met een LSTM-achtige[33] laag. Dat laat bv. toe om extra informatie op woordniveau te injecteren.

Compositie van embeddings komt dus niet van nature met simpele bewerkingen, en indien ze wel een gewenste eigenschap van de embeddingruimte is, vergt ze extra trainingkosten. Het is bovendien onduidelijk of embeddings van *non*-compositionele samenstellingen überhaupt iets met die van hun deelwoorden te maken hebben; Yazdani e.a. (2015) baseren zich exact op de aanname dat dat niet zo is. Al bij al heeft de aanpak van Mikolov e.a. (2013b) wel iets: voeg bepaalde (sub)woord(groep)en toe aan het vocabularium alvorens te trainen, opdat het als een onsplitsbare eenheid wordt behandeld. In §1.4 bleken kleine types nodig in V, maar de conclusie lijkt nu dat er best ook enkele zeer grote types in V zitten – en zoals we in § 2.4.1 zagen, is er een bekende, eindige verzameling lexicaliseringen in elke taal.

Het succes van gecontextualiseerde embeddings doet anderzijds vermoeden dat het uiteindelijk misschien helemaal niet nodig is om een woord of concept in isolement te kunnen voorstellen. LM's verwerken (NLU) of produceren (NLG) volzinnen, waardoor een embedding altijd wordt bijgestaan door omringende embeddings. Voor verdere inferentie met zo'n rij embeddings is de kunst dan *niet* om ze te combineren tot één vector, maar om ze juist als rij te verwerken. Zoals gemotiveerd in §1.4 zal een goede tokeniser voor minder verwarring zorgen en die verwerking zo toelaten met minder parameters.

2.5 BPE-varianten en BPE-alternatieven

BPE is ondertussen niet meer het enige subwoordtokenisatiealgoritme. Er zijn in de loop der tijd varianten vertrekkend van Algoritme 2.1 verschenen in de literatuur, alsook minder bekende alternatieven die een volledig andere aanpak bieden.

Het merendeel van de bronnen die hieronder aan bod komen, zijn reeds vermeld in de overzichtspaper van Mielke e.a. (2021), maar enkel bij wijze van namedropping.[34] Ik tracht om de werking van de respectieve algoritmes vollediger toe te lichten.

[33] Ze gebruiken een iets simpelere variant die nog steeds stabiel is, nl. een *gated recurrent unit (GRU)*.
[34] Hun paper besteedt nl. reeds dubbel zoveel plaats aan een bespreking van pre-tokenisatie.

2.5.1 Algemeen framework

2.5.1.1 Notatie

Het corpus $\mathcal{D}$ bestaat uit zinnen s die uit woorden w bestaan, en na pre-tokenisatie met een gegeven tokeniser bestaan die uit tokens t, elk van een type $\mathfrak{t}$. Daar komt nog bij dat de tokeniser (d.w.z. het vocabularium en de segmentatie van elk woord) iteratief verandert, en nog erger, dat we soms willen spreken over de hypothetische situatie *moest de tokeniser de volgende iteratie op zekere wijze veranderd zijn*. Notatie als "$x \in \mathcal{D}$" en "$C_{\mathcal{D}}(x)$" schiet dus tekort. In wat volgt combineer ik de conventies van Kudo (2018) en Vilar en Federico (2021), volledig opgelijst in Appendix A.

2.5.1.2 Levenscyclus van een tokeniser

In §2.1.2 besprak ik de twee fases in de levenscyclus van een BPE-tokeniser: een leerfase om subwoorden en merges te verzamelen uit het corpus, en een gebruiksfase om aangeleverde woorden op te splitsen. Dat mentale model is echter te nauw om de werking van andere tokenisers, bv. die van He e.a. (2020), te begrijpen. Bovendien is er geen gestandaardiseerde terminologie waarmee naar de levensfases van een tokeniser verwezen kan worden, en zou men die kunnen verwarren met de fases en terminologie van het NLP-model dat erachter volgt.

Vandaar leidde ik het volgende algemenere schema af om te verstaan onder "tokenisatie". Aangezien er in MT een parallel corpus gebruikt wordt, geef ik extra uitleg in die context, en gezien hun decoders op CLM's lijken, licht ik ook die toe.

1. *Vocabularisatie*:[35] leer het tokeniservocabularium.
2. *Inferisatie*: leer de segmentatieregels van de tokeniser.
3. *Corpussegmentatie*: pas de segmentatieregels toe op het traincorpus.

   MT: er kan een aparte tokeniser zijn voor de bron- en doelzijde van het parallelle corpus, en bovendien kan de segmentatie van de ene zijde afhangen van die van de andere. Na deze stap is de doeltaaltokeniser nutteloos.
4. *Train* het NLP-model op het getokeniseerd corpus.

   MT: decodeer één token per keer, waarbij het volgende token niet wordt geconditioneerd op de voorbije *voorspelde* tokens, maar op de *correcte* tokens volgens het doelcorpus (*teacher forcing*).

   CLM: idem.
5. *Invoersegmentatie*: pas de segmentatieregels toe op een invoerstring, die t.o.v. de voorbeelden in het trainingcorpus slechts beperkte informatie bevat.

   MT: de string is de volledige brontaalzin om te vertalen. De vertaling is onbekend en wordt niet door de tokeniser behandeld.

   CLM: de string is het begin van de context die het CLM dient te vervolledigen. De vervollediging is onbekend en wordt niet door de tokeniser behandeld, zelfs al gebruikt het CLM die als bijkomende context.
6. *Infereer* met het NLP-model.

   MT: decodeer een volledige sequentie (tot een eindtoken), waarbij nu wel geconditioneerd wordt op voorbije voorspellingen. Gebruik *beam search*

voor stabiliteit. De uitvoer moet niet worden gesegmenteerd omdat decodering al op tokenniveau gebeurt, en dus impliciet segmentatie nabootst.

CLM: idem.

De "leerfase" van BPE doet tegelijkertijd aan vocabularisatie (resulterend in V) en inferisatie (resulterend in M). Dat is niet zo voor alle algoritmes.

2.5.1.3 Taxonomie

Er zijn veel eigenschappen op basis waarvan we een taxonomie van tokenisers (of hun papers) kunnen opstellen. Mijns inziens is de belangrijkste eigenschap de manier waarop ze gebruikt kunnen worden: er zijn tokenisers die van toepassing zijn op zowel taalmodellen als vertaalmodellen, tokenisers die alleen voor vertaalmodellen gemaakt zijn, en tot slot systemen die het concept van tokens volledig verwerpen en getraind worden samen met het NLP-model. Tabel 2.1 toont enkele andere eigenschappen.

Tokeniser	Bron	F	N	Pr
WordPiece	Schuster e.a. (2012)	B		
BPE	Sennrich e.a. (2016)	B		
CSCS-BPE	Huck e.a. (2017)	B	×	
DLG'-BPE	Wu e.a. (2018)	B		
BPE-dropout	Provilkov e.a. (2020)	B		×
S-BPE	Vilar e.a. (2021)	B		
M-BPE	Banerjee e.a. (2018)	B/M		
Morfessor Baseline	Creutz e.a. (2002)	M		
Morfessor FlatCat	Grönroos e.a. (2014)	M		
LMVR	Ataman e.a. (2017)	M		
Morfessor EM+P	Grönroos e.a. (2020)	M/T		
ULM	Kudo (2018)	T		×
CharacterBERT	El Boukkouri e.a. (2020)	K		
CANINE	Clark e.a. (2022)	K		
Charformer	Tay e.a. (2022)	K		
DPE	He e.a. (2020)	S	×	

Tabel 2.1 – Taxonomie van tokenisers. F = framework (Bottom-up, Top-down, Morfessor, Karakters, Segmenteel); N = alleen bruikbaar voor NMT; Pr = probabilistisch (i.e. de tokeniser produceert voor één zin meerdere segmentaties).

2.5.2 Altijd toepasselijk

2.5.2.1 Varianten op BPE-metriek

DLG'-BPE Wu en Zhao (2018) merken op dat het abstraheren van de metriek op lijn 7 van Algoritme 2.1 een simpele manier is om varianten van BPE te creëren. BPE merget de subwoordtypes die tezamen het frequentst voorkomen, m.a.w. het paar $x, y \in V$ met hoogste $C_{\mathcal{D},\theta}(x, y)$. Wu en Zhao stellen zelf vijf andere metrieken voor om de deftigheid van een paar in te schatten; alle worden gemeten door de huidige tokeniser θ tijdelijk uit te breiden met xy.[36] Twee daarvan, AV en DLG, komen uit hun eigen vakgebied van *Chinesewoordensegmentatie (CWS)* – Chinese zinnen, die uit karakters

[35] Deze naam is uitgevonden door Xu e.a. (2021).

[36] Die tokeniser noem ik in deze sectie $\theta' = \theta \cup \{xy\}$ om notatie te verlichten. Dat volstaat als het altijd over één paar (x, y) gaat, maar anders is θ' uiteraard te vaag.

zonder spaties bestaan, opsplitsen in woorden, analoog met het splitsen van woorden in morfemen – en de andere drie vertrekken vanuit de factor

$$F(xy) = \sum_{w \in W_{\mathcal{D}}} \mathbb{I}\{C_{\theta'}(xy, w) > 0\}\, C_{\mathcal{D}}(w) \tag{2.8}$$

vermenigvuldigd met een modulatiefactor van ofwel de oude BPE-metriek, ofwel AV, ofwel DLG. De gemoduleerde metrieken heten resp. FRQ', AV' en DLG'.

De eerste nieuwe metriek heet *accessor variety (AV)*. De AV van een gemerged paar xy is ofwel het aantal types dat er links van verschijnt in $\mathcal{D}$, ofwel rechts (de kleinste van beide wordt gekozen), genoteerd als

$$\mathrm{AV}(xy) = \min\{|L_{\mathcal{D},\theta'}(xy)|, |R_{\mathcal{D},\theta'}(xy)|\} \tag{2.9}$$

met de gedachte dat een subwoord xy een goede alleenstaande eenheid is als de tokens er links en rechts niet de hele tijd dezelfde zijn.[37] Mijns inziens is er echter een valkuil bij het gebruik van AV-BPE: stel dat twee subwoordtypes x en y mooi met morfemen overeenkomen, dan zouden de types links en rechts van x en y dus zeer gevarieerd moeten zijn (ze hebben apart een hoge AV). Echter, dan is er ook veel kans dat elk voorkomen van het paar (x, y) zo'n variëteit ziet, en dus een hoge AV krijgt, waardoor morfemen makkelijk gaan samenklitten.

De tweede nieuwe metriek heet *description length gain (DLG)* en is gebaseerd op de *shannon-fanolengte (SFL)* van een getokeniseerd corpus, afkomstig uit de informatietheorie. Stel, we zetten de token-id's van alle zinnen in het corpus achter elkaar, gevolgd door een komma en vervolgens alle id's van de paren in M, ook gescheiden door komma's. Dat maakt het corpus verliesloos gecomprimeerd, aangezien we de karakters van het corpus kunnen terugkrijgen door recursief id i te vervangen door de twee tokens na de i'de komma.[38] Noem de resulterende sequentie van tokens X. De SFL van $\mathcal{D}$ is de entropie van de informatiebron die X produceerde, maal de lengte:

$$\mathrm{SFL}(\mathcal{D}) = |X|\, \mathcal{H}(X) = |X| \sum_{t \in V \cup \{,\}} -\frac{C_X(t)}{|X|} \log_2\left(\frac{C_X(t)}{|X|}\right). \tag{2.10}$$

Een extra merge (x, y) uitvoeren betekent dat we dat id-paar achteraan X plakken met een komma ervoor, en het paar vervangen door de id $|M|$ in de rest van X. Het resultaat is normaal kort genoeg om de SFL te verlagen, en in dat geval is het verschil $\mathrm{SFL}(\mathcal{D}) - \mathrm{SFL}(\mathcal{D}_{xy})$ positief: dat is de DLG van (x, y). De BPE-variant die Vgl. 2.8 daarmee vermenigvuldigt, noemen Wu en Zhao *DLG'-BPE*. Hun bahdanau-architectuur bereikt een hogere BLEU met DLG'-BPE dan de andere metrieken, en zeker met Chinese brontekst (maar ook de Engelse doeltekst baat bij DLG'-BPE).

Het zou inefficiënt zijn moest de argmax op lijn 7 de bovenstaande berekeningen voor elk typepaar moeten uitvoeren, maar gelukkig stelden Kit en Wilks (1999) reeds een efficiënte methode voor die de berekeningen van $\mathrm{SFL}(\mathcal{D})$ recycleert, en bovendien houden zij rekening met het algemenere geval waarin men per merge kan kiezen hoeveel types er gemerged worden (i.p.v. altijd 2 zoals in BPE). De auteurs stellen vandaar ook een segmentatiealgoritme voor dat een heel corpus (een lange string inclusief spaties) optimaal segmenteert volgens DLG d.m.v. een viterbialgoritme (zie Appendix B voor een opfrisser daarover). Zhao en Kit (2008) veralgemenen dat naar andere metrieken voor CWS.

[37] De uitvinders van AV, Feng e.a. (2004), tellen bij L en R ook het aantal keer dat xy aan het begin resp. einde van een Chinese zin staat. Het is onduidelijk of Wu en Zhao (2018) dat nu voor het begin resp. einde van een woord doen.

[38] Aannemend dat de 0'de merge in M type-id 0 heeft, de 1'ste merge type-id 1 etc ... waarbij letters in het alfabet niet op te zoeken zijn.

S-BPE Bij de berekening van entropie in Vgl. 2.10 was de aanname dat de informatiebron elk token genereerde door het onafhankelijk uit een categorische verdeling te samplen. Vilar en Federico (2021) werken met dezelfde aanname, maar bekijken het probleem vanuit Bayesiaanse statistiek. Stel alle vrijheidsgraden van de tokeniser voor met $\boldsymbol{\Theta}$, dan willen we de tokeniser $\boldsymbol{\theta}$ vinden met de hoogste kans gegeven het corpus, die dan het *maximum a posteriori (MAP)* heet. Volgens de wet van Bayes:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \frac{P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} = \arg\max_{\boldsymbol{\theta}} P(\mathcal{D} \mid \boldsymbol{\theta})P(\boldsymbol{\theta}). \tag{2.11}$$

Daarin heet $P(\boldsymbol{\theta} \mid \mathcal{D})$ de *posterior* van de tokeniser $\boldsymbol{\Theta} = \boldsymbol{\theta}$, heet $P(\mathcal{D} \mid \boldsymbol{\theta})$ de *aannemelijkheid* (Eng.: *likelihood*) van het corpus indien het met $\boldsymbol{\theta}$ gesegmenteerd wordt, en heet $P(\boldsymbol{\theta})$ de *prior* die een vooroordeel over de goedheid van bepaalde tokenisers voorstelt. Indien we geen voorkeur hebben a priori, is $P(\boldsymbol{\theta})$ constant en blijft alleen de likelihood over. MAP wordt dan *maximum likelihood (ML)*:

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} P(\mathcal{D} \mid \boldsymbol{\theta}). \tag{2.12}$$

Vilar en Federico definiëren zo de likelihood van het gesegmenteerde corpus als de kans van een opeenvolging van onafhankelijke tokens (die dan *unigrams* heten):

$$P(\mathcal{D} \mid \boldsymbol{\Theta}) = \prod_{t \in \mathcal{T}_{\mathcal{D},\Theta}} P(t) = \prod_{t \in V_\Theta} P(t)^{C_{\mathcal{D},\Theta}(t)}. \tag{2.13}$$

Wiskundig is het vaak voordeliger om $\mathcal{L}(\mathcal{D};\theta) = \ln P(\mathcal{D} \mid \boldsymbol{\theta})$, de *log-likelihood*, te maximaliseren, wat equivalent is gezien $\ln(x)$ monotoon stijgend is:

$$\mathcal{L}(\mathcal{D};\theta) = \ln P(\mathcal{D} \mid \boldsymbol{\Theta}) = \sum_{t \in V_\Theta} C_{\mathcal{D}}(t) \ln P(t) \tag{2.14}$$

Vilar en Federico definiëren de beste merge als die waardoor de likelihood $\mathcal{L}(\mathcal{D})$ het meeste stijgt. Ze tonen formeel aan dat de stijging veroorzaakt door een merge $x\,y \to xy$ als ondergrens

$$C_{\mathcal{D},\theta'}(xy) \ln\left(\frac{P(xy)}{P(x)P(y)}\right) \tag{2.15}$$

heeft, hetgeen ze in lijn 7 invullen. De typekansen $P(t)$ worden berekend in het corpus dat met θ' gesegmenteerd is, opdat het type xy überhaupt voorkomt, en veranderen dus ook na elke merge. Hun waarde bepalen we uit dat tijdelijke corpus a.d.h.v. een *maximum likelihood estimator (MLE)*: met de methode van lagrangevermenigvuldigers maximaliseren we Vgl. 2.14 i.f.v. $P(t) = p_t$, onder voorwaarde dat $\sum_{t \in V_{\theta'}} p_t = 1$, als volgt.

$$\begin{aligned} \forall t \in V_{\theta'} : \frac{\partial}{\partial p_t}\left(\sum_{t \in V_{\theta'}} C_{\mathcal{D},\theta'}(t) \ln p_t - \lambda\left(\sum_{t \in V_{\theta'}} p_t - 1\right)\right) &= 0 \\ C_{\mathcal{D},\theta'}(t)\,\frac{\partial}{\partial p_t} \ln p_t - \lambda &= 0 \\ p_t &= C_{\mathcal{D},\theta'}(t)/\lambda \end{aligned} \tag{2.16}$$

... waarin $\forall t \in V$ dezelfde $\lambda \in \mathbb{R}$ wordt gebruikt, en dus wordt de beperking

$$\begin{aligned} 1 = \sum_{t \in V_{\theta'}} p_t = \sum_{t \in V_{\theta'}} \frac{C_{\mathcal{D},\theta'}(t)}{\lambda} = \frac{1}{\lambda} \sum_{t \in V_{\theta'}} C_{\mathcal{D},\theta'}(t) = |\mathcal{T}_{\mathcal{D},\theta'}|/\lambda \\ \text{en dus } \begin{cases} \lambda = |\mathcal{T}_{\mathcal{D},\theta'}| \\ p_t = C_{\mathcal{D},\theta'}(t)/|\mathcal{T}_{\mathcal{D},\theta'}| \quad (\forall t \in V) \end{cases} \end{aligned} \tag{2.17}$$

... dus kunnen we gewoon de tokens van een type tellen als we zijn kans willen weten. De tokeniser die Vgl. 2.15 gebruikt als metriek en voor de rest hetzelfde als BPE werkt, heet *statistische BPE (S-BPE)*.

De auteurs wijzen erop dat die metriek sterk lijkt op *puntgewijze mutuele informatie (PMI)*, die normaal tussen twee woorden berekend wordt:

$$\mathrm{PMI}(w_1, w_2) = \frac{P(w_1, w_2)}{P(w_1)P(w_2)} \overset{\text{MLE}}{\propto} \frac{C_{\mathcal{D}}(w_1 w_2)}{C_{\mathcal{D}}(w_1)C_{\mathcal{D}}(w_2)} \tag{2.18}$$

Die laatste uitdrukking gebruiken Mikolov e.a. (2013b) en Kumar en Thawani (2022) om MWE's te vinden die meer samen dan apart horen, en Berton e.a. (1996) deden hetzelfde maar na de wortel van de noemer te nemen. PMI is hoogst wanneer twee (sub)woorden veel naast elkaar voorkomen en weinig daarbuiten, waardoor het i.t.t. AV wel een duidelijk signaal geeft wanneer we te veel mergen.

WordPiece Hoewel Sennrich e.a. (2016) er niets over vermeldden, is het ondertussen welbekend dat Schuster en Nakajima (2012) de idee achter BPE reeds eerder beschreven voor hun *WordPieceModel (WPM)*. Volgens HuggingFace is WPM simpelweg BPE met PMI (Vgl. 2.18) als metriek. De werkelijkheid is genuanceerder: Schuster en Nakajima beschrijven een propriëtair spraakherkenningssysteem van Google uit 2009, waarbij ze zonder exacte details een algoritme uiteenzetten dat weliswaar net als BPE vertrekt van karaktertypes en dan bottom-upgewijs paren merget, maar de metriek geven ze bv. nooit prijs. (Toch is WPM in gebruik gebleven, bv. in Google Translate (Wu e.a., 2016) en tevens in BERT omdat het onderzoek van Devlin e.a. (2019) bij Google plaatsvond en zij dus toegang hadden tot de WPM-implementatie.)

In woorden beschrijven Schuster en Nakajima hun metriek op exact dezelfde manier als Vilar en Federico (2021), nl. de toename in likelihood van het getokeniseerd corpus, alleen berekenen ze die niet met Vgl. 2.13 maar met een N-gram-LM, vermoedelijk als

$$P(\mathcal{D} \mid \Theta) = \prod_{i=1}^{|\mathcal{T}_{\mathcal{D},\Theta}|} P(t_i \mid t_{i-1}, \ldots, t_{i-N+1}). \tag{2.19}$$

De factoren volgen weer uit frequentiegebaseerde MLE's. Let wel dat WPM i.t.t. S-BPE geen efficiëntere ondergrens (Vgl. 2.15) gebruikt.

Daar stoppen de verschillen met de oorspronkelijke WPM en BPE niet. Zo is er niet één manier om spaties voor te stellen (cfr. de EOW op pagina 16), maar drie: aan het begin van een subwoord, aan het einde, of de twee samen. Het vocabularium is dus dubbel zo opgezwollen, met bv. de subwoorden {. . . , -en-, -en, en-, en, . . .}. Bovendien start WPM met een Unicode- i.p.v. ASCII-alfabet omdat het spraaksysteem voor Japans en Koreaans moest dienen. Door de grote $|V|$ en het gebruik van een 5-gram-LM voor de metriek duurt de leerfase veel langer, waartoe de auteurs versnellingstechnieken toepassen (bv. hergebruik van de likelihood van het huidige N-gram-model, subwoordparen overslaan die niet in het corpus voorkomen ...). Tot slot beweert HuggingFace dat segmentatie anders gebeurt in WPM dan in BPE:

> Tokenization differs in WordPiece and BPE in that WordPiece only saves the final vocabulary, not the merge rules learned. Starting from the word to tokenize, WordPiece finds the longest subword [at the start of the word] that is in the vocabulary, then splits on it.

Nochtans ontkennen Schuster en Nakajima dat soort *greedy* segmentatie van links naar rechts, daar ze wel degelijk een tokenisatieboom zoals Figuur D.7 beschrijven:

> As the segmentation builds an inverse binary tree of pairs starting from the basic character symbols, the segmentation itself does not need any Dynamic Programming or other search procedures [...] in linear time with respect to the length of the sentence.

2.5.2.2 Varianten op het gebruik van de BPE-tokeniser

BBPE Het alfabet van WPM is bijzonder groot, gezien het niet alleen alle ASCII-karakters bevat, maar ook 22 000 Japanse en 11 000 Koreaanse tekens (Schuster en Nakajima, 2012). Om teksten uit alle talen met speciale tekens (bv. cyrillisch, Grieks, Arabisch ...) te ondersteunen beslissen Radford e.a. (2019) daarom om BPE niet op karakters – ook wel *codepunten* (Eng.: *code points*) – toe te passen, maar op de rauwe UTF-8-bytes waaruit ze bestaan (zoals de voorbeeldsequentie op pagina 4). Een byte heeft slechts 256 mogelijke waarden, dus het alfabet is even klein en beperkt, met voldoende ruimte om de relevante UTF-8-coderingen te leren. De pretokeniser van Radford e.a. komt in feite neer op `bytes(corpus, "utf-8")` in Python, zonder bv. eerst leestekens apart te plaatsen. In plaats daarvan verbieden ze merges tussen aparte Unicode-categorieën, waaronder letters, leestekens, wiskundige tekens en witruimte.[39]

ASCII-karakters bestaan in UTF-8 elk uit 1 unieke byte, dus voor ASCII-teksten is er geen verschil met ASCII-gebaseerde BPE. UTF-8 is bovendien *zelfsynchroniserend*: niet-ASCII-karakters bestaan uit 2, 3 of 4 bytes, waarvan er geen enkele een ASCII-byte is en de eerste byte telkens uitsluitend karakters kan inleiden in één van de drie klassen. Die eigenschap zorgt dat een sequentie nooit ambigu te interpreteren is, en karaktergrenzen makkelijk te interpreteren zijn. Niettemin vertrekken Radford e.a. vanaf enkelvoudige bytes, dus zitten er gegarandeerd subwoorden in V die op zichzelf geen geldig Unicode-codepunt zijn: Wang e.a. (2019) stellen een viterbialgoritme voor (gelijkend op dat in §B.2) om bij inferentie zoveel mogelijk geldige codepunten te halen uit een geproduceerde sequentie van subwoorden, en benoemen die variant tot *byte-gebaseerde BPE (BBPE)*.

Radford e.a. (2019) gebruiken BBPE voor GPT-2, en Liu e.a. (2019) voor RoBERTa:

> [...] with [BBPE] achieving slightly worse end-task performance on some tasks. Nevertheless, we believe the advantages of a universal encoding scheme outweighs the minor degredation in performance [...]

iBPE Zoals reeds besproken hebben grotere subwoordtypes (die gevormd worden bij hogere $|V|$, zie Figuur D.2) een lagere frequentie dan kleinere subwoordtypes, waardoor er minder trainingdata is die toelaat om te leren hoe ze passend te gebruiken. Echter, volgens de GH (§1.4) kunnen we het gebruik van een groot subwoord deels afleiden uit het gebruik van de subwoorden die erin vervat zitten. Eén manier om kennis over te dragen is om het model eerst uitsluitend te trainen op een corpus met kleine subwoorden, en daarna pas grotere subwoorden toe te laten die geïnformeerd worden op basis van wat er tot dan toe geleerd is.

Normaal gezien heeft BPE één leerfase, waarna de tokeniser ($\Theta = (V, M)$) voorgoed bevroren wordt en de corpussegmentatie eenmalig uitvoert. Het navolgende LM of NMT-model ziet dientengevolge voor elke zin altijd dezelfde segmentatie. Salesky e.a. (2018) alterneren in plaats daarvan tussen de leerfase van BPE, corpussegmentatie, en het trainen van hun bahdanausysteem. De leerfase doet V groeien met 10 000 subwoordtypes

[39] Ze schrijven dat ze voor spaties een uitzondering maken, maar het is onduidelijk of ze daarmee in de conventie belanden van BPE (al dan niet een EOW achter een subwoordtype) of van WPM (al dan niet een EOW ervoor en al dan niet erachter). Aannemelijk blijven ze net als BPE nog wel woordgrenzen respecteren (i.e. spaties kunnen opgenomen worden in subwoordtypes, maar niet in het midden).

per keer, waarbij de embeddings van die bijkomende types worden geïnitialiseerd met een AE zoals in §2.4.2.1 werd toegelicht. Vervolgens worden het training- en tuningcorpus opnieuw gesegmenteerd. Tot slot trainen ze het NMT-model gedurende minstens een gegeven aantal *epochs* (elke epoch doorloopt het trainingcorpus eenmaal), waarna ze weer pauzeren eens het modelverlies stagneert over het tuningcorpus. Die *incrementele BPE (iBPE)* heeft m.a.w. ook geen nood aan de limiet τ in Algoritme 2.1, gezien het model dynamisch bepaalt wanneer een nieuwe lading subwoorden gepast is (zie §4.12).

2.5.2.3 Probabilistisch

Kudo (2018) beargumenteert in het eerste deel van zijn paper dat een tokeniser voor één zin meerdere mogelijke segmentaties moet kunnen aanbieden, en moet toelaten daartussen te samplen. Net als He e.a. (2020) wijst hij erop dat men de segmentatie van een zin s eigenlijk moet zien als een toevalsvariabele over $2^{|s|-1}$ uitkomsten (cfr. §2.4.1), en dat BPE dus een extreem, deterministisch geval is waarbij dier kansmassa 1 is voor de BPE-segmentatie en 0 daarbuiten.

Volgens Kudo moet men in de verliesfunctie van een model niet één log-likelihoodwaarde $\ln P(x_{\text{BPE}})$ minimaliseren per voorbeeld x in het trainingcorpus, maar de verwachte waarde over alle segmentaties, $\mathbb{E}_x[\ln P(x)]$. Die is te benaderen met de gemiddelde waarde over k gesamplede segmentaties, hetgeen hij *subwoordregularisatie* noemt. Met genoeg epochs is $k = 1$ voldoende (vergelijkbaar met *stochastic gradient descent (SGD)*) om voor dezelfde zin genoeg alternatieve segmentaties tegen te komen, waardoor het model het verband tussen subwoorden beter leert.

BPE-dropout We zagen enkele varianten die in Algoritme 2.1 alleen de metriek van de leerfase aanpasten. We kunnen ook de gebruiksfase aanpassen en de leerfase met rust laten. Provilkov e.a. (2020) stellen de simpelste manier voor om subwoordregularisatie in BPE te integreren: elke iteratie van het segmenteren negeren ze op lijn 18 met vaste kans p een geldige merge. Aangezien een genegeerde merge de volgende iteratie wel weer beschikbaar kan zijn, moeten de prioritaire merges eerst weer gecheckt worden, waardoor de aanpassing iets complexer is; zie de gekleurde lijnen in Algoritme 2.2. Het resultaat heet *BPE-dropout*. Let wel: BPE-dropout vindt alleen plaats tijdens corpussegmentatie. De leerfase (vocabularisatie en inferisatie) en invoersegmentatie worden van BPE overgenomen (m.a.w. met $p = 0$).

Algoritme 2.2 Pseudocode BPE-dropout

```
1:  function GEBRUIKBPEDROPOUT(V, M, w, p)
2:      w ← SPLITSINLETTERS(w) + [EOW]
3:      i ← 0
4:      while i < |M| do
5:          (x, y) ← M[i]
6:          i ← i + 1
7:          if x_y ∈ w then
8:              if RANDOM(0,1) < p then
9:                  w ← VERVANG1(w, x_y, xy)
10:                 i ← 0
11:     return [INDEX(V, t) | t ∈ w]
```

Hoewel niemand BPE-dropout met iBPE associeert (voor zover ik weet), zijn er toch enkele gelijkenissen te vinden: beide vinden alleen tijdens training plaats, beide zorgen ervoor dat dezelfde zin doorheen de epochs andere segmentaties kan krijgen, en beide

prioriteren een bepaald deel van de BPE-mergeboom van een woord – alleen prioriteert iBPE steeds grotere subwoorden, terwijl de kans op een subwoord in BPE-dropout juist geometrisch dáált met zijn lengte.

ULM Omdat BPE-dropout nog niet uitgevonden was toen Kudo (2018) subwoordregularisatie beschreef, stelt hij in het tweede deel van zijn paper ook een nieuwe probabilistische tokeniser op. Hij vertrekt vanuit de aanname dat in eender welke segmentatie van een zin de tokens onafhankelijk zijn van elkaar, en dus net als S-BPE (Vgl. 2.13) de taal modelleert met de unigram-aanname. Zijn tokeniser noemt hij vandaar ook letterlijk *unigram-LM (ULM)*.[40]

Wel anders is dat het corpus nu geen vaste segmentatie meer heeft: we moeten overheen alle segmentaties van dezelfde invoer kunnen samplen, en dus moet de likelihood die ook allemaal meetellen i.p.v. het corpus als één rij van tokens te beschouwen. Stel dat $\mathcal{S}_V$ alle segmentaties van een zin geeft die te vormen zijn met vocabularium V, dan is

$$P(\mathcal{D} \mid \Theta) = \prod_{s \in \mathcal{D}} P(s \mid \Theta) = \prod_{s \in \mathcal{D}} \sum_{\zeta \in \mathcal{S}_V(s)} P(\zeta, s \mid \Theta) = \prod_{s \in \mathcal{D}} \sum_{\zeta \in \mathcal{S}_V(s)} \prod_{t \in \zeta} p_t \tag{2.20}$$

...waarbij de tokeniser Θ bestaat uit enerzijds V en anderzijds de tokenkansen $\vec{p} = (p_1, \ldots, p_{|V|})$, die net als in S-BPE som 1 hebben. Net als S-BPE en WPM quoteert ULM een subwoord op basis van een verandering in likelihood, maar in tegenstelling tot die twee werkt ULM top-down i.p.v. bottom-up: waar S-BPE en WPM met een kleine V starten en telkens het subwoord aan V toevoegen dat de likelihood het meest doet stijgen, start ULM juist met een gigantische V om dan telkens de subwoorden uit V te verwijderen die de likelihood op die manier het minst doen dalen (en dus geen cruciale rol spelen).

ULM heeft ook een leerfase en een gebruiksfase. De leerfase alterneert tussen vocabularisatie (het leren van V) en inferisatie (het leren van $\vec{p}$). Het initiële vocabularium bestaat uit een arbitraire hoeveelheid substrings in het trainingcorpus, zoals bv. de 1 miljoen meest frequente (Grönroos e.a., 2020). Vocabularisatie gebruikt de daling in Vgl. 2.20, zoals reeds vermeld. Algoritme 2.3 toont hoe dat kan zonder het hele corpus opnieuw te segmenteren per verwijderd type, en op een numeriek stabiele manier, nl. door optelling van logaritmes i.p.v. producten van kansen in $[0, 1]$. Vgl. 2.20 geeft een log-likelihood

$$\mathcal{L}(\mathcal{D}; \Theta) = \ln P(\mathcal{D} \mid \Theta) = \ln \prod_{s \in \mathcal{D}} \sum_{\zeta \in \mathcal{S}_V(s)} P(\zeta, s \mid \Theta) = \sum_{s \in \mathcal{D}} \ln \left(\sum_{\zeta \in \mathcal{S}_V(s)} P(\zeta, s \mid \Theta) \right) \tag{2.21}$$

en dus kunnen we de afname in likelihood meten per zin en bij een globale teller optellen. Hoe veranderen de segmentaties van een zin s als we type t verwijderen uit V? Alle segmentaties blijven mogelijk, behalve die met t, die onmogelijk worden. Het verschil in de likelihood van s tussen de voor- en de nasituatie is dan

$$\begin{aligned} \Delta\mathcal{L}(s; \Theta)[t] &= \ln \left(\sum_{\zeta \in \mathcal{S}_V(s)} P(\zeta, s \mid \Theta) \right) - \ln \left(\sum_{\zeta \in \mathcal{S}_V(s) \mid t \notin \zeta} P(\zeta, s \mid \Theta) \right) \\ &= \ln(P_s) - \ln \left(P_s - \sum_{\zeta \in \mathcal{S}_V(s) \mid t \in \zeta} P(\zeta, s \mid \Theta) \right) \end{aligned} \tag{2.22}$$

[40] Die eigenschap is echter niet uniek – S-BPE (§2.5.2.1) en Morfessor (§2.5.2.4) modelleren de likelihood eveneens met een unigram-LM. Er zijn dus beter typerende benamingen te vinden, bv. "probabilistische EM-gebaseerde top-down-tokenisatie".

met P_s de kansmassa van alle segmentaties van s tezamen en $P(\zeta, s \mid \Theta)$ het product op lijn 11. Opdat we elke zin in het corpus slechts eenmalig zouden zien en $\mathcal{S}_V$ eenmalig zouden oproepen, kunnen we die laatste sommatie in parallel bijhouden voor elke $t \in V$ bij het itereren overheen de segmentaties van s. Dat is wat Algoritme 2.3 doet.

Inferisatie wordt gedaan met een iteratief *EM-algoritme* dat Kudo (2018) niet verder toelicht; Grönroos e.a. (2020) constateren hetzelfde, en wijzen erop dat de officiële implementatie van ULM in het SentencePiece-pakket anders is dan de paper doet vermoeden en verre van triviaal is. Ik ga er niet verder op in.
Desalniettemin wijs ik er graag op dat de HuggingFace-documentatie over ULM incorrect is en niet strookt met hun eigen implementatie: de documentatie beweert dat er geen EM gebruikt wordt, en dat het schatten van $\vec{p}$ gebeurt door op voorhand elke mogelijke string in $\mathcal{D}$ te tellen, en na elke iteratie van de leerfase die aantallen op te zoeken voor de strings die nog in V zitten en daarover te normaliseren. Die aanpak zorgt ervoor dat de kansen van subwoorden altijd dezelfde volgorde hebben, zelfs al verdwijnen er veel segmentaties waarin hoogfrequente strings als token voorkomen. Over vocabularisatie beweert de documentatie bovendien dat Vgl. 2.20 zogezegd niet over alle segmentaties sommeert, maar slechts over $|\mathcal{S}_V(s)| = 1$, nl. de viterbisegmentatie.

Na de leerfase hebben we een verzameling subwoordtypes V en hun unigram-kansen $\vec{p}$ verkregen. We kunnen daarmee een betrouwbare kansmassa overheen elke segmentatie ζ van een zin berekenen, nl. lijn 11 in Algoritme 2.3. Om een LM of TM te leren kunnen we tijdens corpussegmentatie ter regularisatie rechtstreeks uit die kansmassa samplen, i.t.t. BPE-dropout. Nadien willen we voor inferentie dat de invoerzin deterministisch gesegmenteerd wordt, en zo goed mogelijk: in dat geval zoeken we de segmentatie die lijn 11 maximaliseert, en aangezien ULM de tokens in de zin onafhankelijk acht, kan dat met een viterbialgoritme in kwadratische tijd i.p.v. exponentiële tijd (zie Appendix B).

Bostrom en Durrett (2020) concluderen dat ULM significant veel beter werkt dan BPE. Hun paper werd echter parallel met die van BPE-dropout geschreven, en Amrhein en Sennrich (2021) raden die laatste dan weer aan. Provilkov e.a. (2020) vergelijken zelf met ULM op basis van BLEU alleen, en vinden dat ULM sterker is dan BPE maar zwakker dan BPE-dropout.

2.5.2.4 Morfessor

De geschiedenis van subwoordtokenisatie gaat nog veel verder terug dan WPM. Reeds twintig jaar geleden stelden Creutz en Lagus (2002) een generatief model voor dat later tot een familie van tokenisers is uitgegroeid met de naam *Morfessor* (Creutz en Lagus, 2007).[41] De kern van Morfessor is dat het op MAP (Vgl. 2.11) i.p.v. ML gebaseerd is, waarbij de extra priorfactor $P(\Theta)$ een model is voor het ontstaansproces van het vocabularium zelf. De likelihood $P(\mathcal{D} \mid \Theta)$ is identiek aan die van S-BPE (Vgl. 2.13), en modelleert het ontstaan van het corpus gegeven dat het vocabularium reeds bestaat. De prior van Creutz en Lagus (2002) is veelal gebaseerd op een soort SFL (cfr. §2.5.2.1) uit de informatietheorie, waarbij subwoordtypes met grotere lengte een lagere kans hebben. Dat maakt dat de prior en de likelihood elkaar weerstand bieden: de likelihood is immers een product van tokenkansen – getallen in $[0, 1]$ – en prefereert dus juist dat het corpus met minder (en dus grotere) tokens wordt voorgesteld.

Morfessor Baseline Creutz (2003) stelt een volledige generatieve procedure op voor het vocabularium – d.w.z. een fictief verhaaltje op basis waarvan we voor een daadwer-

[41] Er zijn ongeveer vijf varianten: Morfessor Baseline, Morfessor Cat-ML, Morfessor Cat-MAP, Morfessor FlatCat, en Morfessor EM+Prune.

Algoritme 2.3 Pseudocode ULM

function LEERULM($\mathcal{D}$, τ, η)
  $V \leftarrow$ ALLESUBSTRINGS($\mathcal{D}$)
  $\vec{p} = (p_1, \ldots, p_{|V|}) \leftarrow (\frac{1}{|V|}, \ldots, \frac{1}{|V|})$
  while $|V| > \tau$ **do**
    $\vec{p} \leftarrow$ HERSCHAT($\mathcal{D}$, V, $\vec{p}$)
    $\Delta\vec{\mathcal{L}} \leftarrow \vec{0}$ ▷ Totale log-likelihoodafname per type
    for $s \in \mathcal{D}$ **do**
      $P_s \leftarrow 0$
      $\Delta\vec{P}_s \leftarrow \vec{0}$ ▷ Kansafname in s per type
      for $\zeta \in$ SEGMENTATIES(s, V) **do**
        $P_\zeta \leftarrow \prod_{t \in \zeta} p_t$ ▷ Kans van de zinsegmentatie...
        $P_s \leftarrow P_s + P_\zeta$
        for $t \in \zeta$ **do**
          $\Delta\vec{P}_s[t] \leftarrow \Delta\vec{P}_s[t] + P_\zeta$ ▷ ...valt weg als t wegvalt.
      $\Delta\vec{\mathcal{L}} \leftarrow \Delta\vec{\mathcal{L}} + \ln(P_s) - \ln(P_s - \Delta\vec{P}_s)$
    $V \leftarrow$ SORTEERAFLOPEND(V, $\Delta\vec{\mathcal{L}}$)
    $V \leftarrow V\,[0 : \eta|V|]$ ▷ Weinig likelihood weggevallen ⇒ niet nodig
  return $(V, \vec{p})$

function GEBRUIKULM(w, V, $\vec{p}$)
  Voorwaartse fase (iteratieve i.p.v. recursieve viterbi):
  $\ell_1 \ldots \ell_{|w|} \leftarrow -\infty$ ▷ Beste score tot nu toe gezien door substring $w\,[1 : i]$
  $t_1 \ldots t_{|w|} \leftarrow \langle \texttt{UNK} \rangle$ ▷ Beste eindtoken tot nu toe voor substring $w\,[1 : i]$
  for $i = 1 \ldots |w|$ **do**
    for $j = i + 1 \ldots |w|$ **do**
      $t \leftarrow w\,[i : j]$
      if $t \in V$ **then**
        $\ell \leftarrow \ell_i + \ln p_t$
        if $\ell > \ell_j$ **then**
          $\ell_j \leftarrow \ell$
          $t_j \leftarrow t$
  Achterwaartse fase (viterbi-tracé):
  $T \leftarrow$ []
  $i \leftarrow |w|$
  while $i > 0$ **do**
    $T \leftarrow [t_i] + T$
    $i \leftarrow i - |t_i|$
  return T

kelijk vocabularium kunnen inschatten hoe plausibel zijn bestaan is. Die gaat als volgt: eerst wordt de grootte $|V|$ van het vocabularium willekeurig gekozen. Dan worden er $|V|$ subwoordtypes gegenereerd: eerst samplet men de lengte uit een gammaverdeling (vergelijkbaar met een poissonverdeling die minder streng is jegens grotere lengtes), en vervolgens samplet men zoveel letters onafhankelijk uit de letterverdeling van de taal in kwestie. Merk op dat dat proces eenzelfde vocabularium kan genereren op $|V|!$ manieren; de volgorde van de subwoorden maakt niet uit. De prior schat tot slot ook hoe plausibel de globale verdeling van de subwoorden over het corpus is.[42] Creutz (2003) stelt voor om de kans van één frequentie af te leiden uit de wet van Zipf, maar omdat dat "onnodig ingewikkeld en onvolledig" is (Creutz en Lagus, 2005b), nemen latere versies van Morfessor een minder informatieve prior: gegeven dat er $|V|$ subwoorden zijn en het corpus $N = |\mathcal{T}_{\mathcal{D},\Theta}|$ tokens bevat, zijn er $\binom{N-1}{|V|-1}$ mogelijke manieren om elk token aan een type toe te wijzen. Als elk van die toewijzingen dezelfde kans heeft om voor te komen, dan hebben ze elk kans $1/\binom{N-1}{|V|-1}$. De volledige prior van *Morfessor Baseline* is zo:

$$P(\Theta) = P(|V|) \cdot |V|! \cdot \prod_{t \in V} \left[P_\Gamma(|t|) \prod_{c \in t} P(c) \right] \cdot \binom{|\mathcal{T}_{\mathcal{D},\Theta}| - 1}{|V| - 1}^{-1} \tag{2.23}$$

... waarin $P(|V|) = p$ een constante is, P_Γ twee hyperparameters heeft (Creutz en Lagus, 2005b) en $P(c)$ een MLE is (het aantal keer dat karakter c in het corpus voorkomt op het aantal karakters in het corpus, gelijkaardig aan Vgl. 2.17).

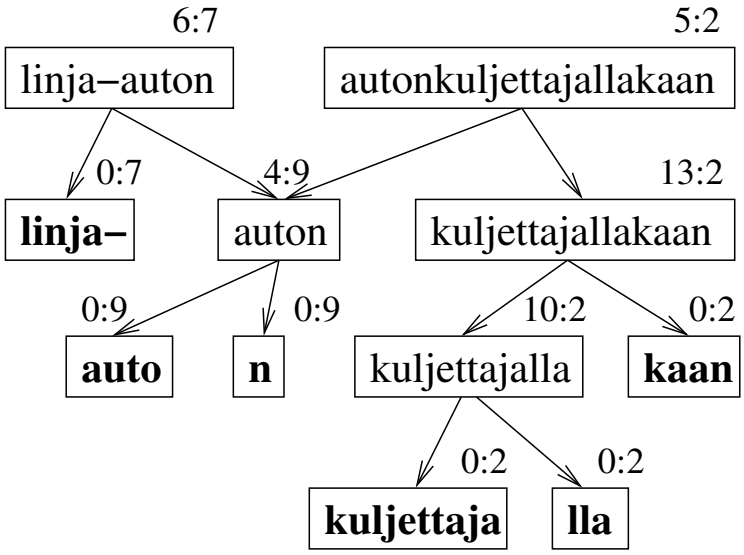
Morfessor Baseline verwerkt $\mathcal{D}$ één woord per keer en zoekt daarbij naar de meest geschikte V. Het nut van de prior is om tijdens dat zoekalgoritme tegendruk tegen de likelihood te geven; de likelihood wil woorden in hun geheel aan V toevoegen, terwijl de prior aanzet tot splitsen in bestaande subwoorden. Vgl. 2.11 is op die manier de metriek die de zoektocht begeleidt, gelijkaardig aan hoe BPE door een metriek geleid wordt. De leerfase houdt een boom bij zoals die in Figuur 2.3, die een mix is tussen de BPE-mergegraaf en een prefixboom: in elke knoop zit een (string, aantal)-paar zoals in een prefixboom, maar ouders bevatten hun kinderen zoals in BPE – alleen bevat V nu louter de bladeren i.p.v. alle knopen, want de totale boom zal uiteindelijk minstens één knoop voor elk woord in $\mathcal{D}$ bevatten. Daardoor zijn het ook alleen veranderingen in de bladeren die Vgl. 2.23 en Vgl. 2.13 verhogen of verlagen.

Algoritme 2.4 geeft een verkorte versie van algoritme 1 van Creutz en Lagus (2005b). De functies VERWIJDER en VOEGTOE werken recursief op de subboom waar de gegeven string de wortel van is, en voor elk blad dat ze tegenkomen verandert de posterior $\mathcal{L}$. Een knoop met aantal 0 verdwijnt uit de boom. Als we met VOEGTOE kinderen toevoegen aan een knoop die er nog geen heeft, dan verwijderen we diens bijdrage aan $\mathcal{L}$ als blad op dat moment.

Creutz en Lagus raden ook aan om het itereren over het corpus periodiek te onderbreken om in plaats daarvan te itereren over een gehallucineerd corpus dat bestaat uit willekeurig gekozen knopen in de boom. Zo krijgen oude woorden de kans om beter geïntegreerd te worden in de nieuwere boom.

Hoewel elk woord w van $\mathcal{D}$ na afloop een knoop heeft in de boom en de bladeren van zijn subboom een geldige segmentatie geven (cfr. Figuur 2.3), gebeurt segmentatie alsnog met het viterbialgoritme van ULM (Algoritme 2.3).

[42] Ter verduidelijking: de likelihood schat de kans om het exacte corpus $\mathcal{T}_{\mathcal{D},\Theta}$ te genereren. Waar het hier over gaat is de kans om in *eender welk corpus* van grootte $|\mathcal{T}_{\mathcal{D},\Theta}|$ de proporties van subwoorden te vinden zoals die in $\mathcal{T}_{\mathcal{D},\Theta}$ te vinden zijn. Stel bv. dat alle subwoorden evenveel gebruikt worden doorheen het corpus, dan weten we dat er iets mankeert aan V gezien echte taal zich zo niet gedraagt.



Figuur 2.3 – Zoekgraaf van enkele Finse Morfessor-subwoordtypes. De bladeren zijn vetgedrukt en zitten in V. Elke knoop is geannoteerd met een splitsingsindex en het aantal voorkomens. Bron: figuur 1 van Creutz en Lagus (2002).

Morfessor Cat-MAP Een groot probleem met unigram-taalmodellen als Vgl. 2.13 is dat een sequentie tokens even waarschijnlijk is in eender welke volgorde. Daardoor begint Morfessor Baseline bv. te hallucineren over suffixmorfemen aan het begin van een woord: in de woorden *swing* en *wings* heeft de *s* duidelijk een andere rol, maar Morfessor Baseline beoordeelt de slechte segmentatie *s*|*wing* alsof ze even goed is als de goede segmentatie *wing*|*s*.

Contextafhankelijkheid treedt pas op in een bigrammodel of *markovketen*: een rij tokens $t_1, t_2, t_3 \ldots$ heeft daarin niet meer de kans $P(t_1)P(t_2)P(t_3)\ldots$, maar wel het eersteordeproduct $P(t_1 \mid \text{start})P(t_2 \mid t_1)P(t_3 \mid t_2)\ldots$ Merk echter op dat het probleem hierboven niet zozeer was dat bepaalde *tokens* elkaar niet mochten volgen, maar dat dat eerder gold voor bepaalde *tokenklassen* – een suffix mag niet voor een stam staan, een stam niet voor een prefix, etc ... Er is m.a.w. een achterliggende markovketen van tokenklassen K die als bijproduct tokens t van die klassen produceert: een *verborgen markovmodel* (Eng.: *hidden Markov model, HMM*). Creutz en Lagus (2005a) vervangen het unigrammodel van likelihood met een HMM die door elk woord loopt:

$$P(\mathcal{D} \mid \Theta) = \prod_{w \in W_\mathcal{D}} \left(P(K_{w,1} \mid K_0) \prod_{i=1}^{|\Theta(w)|} P(K_{w,i+1} \mid K_{w,i}) P(t_{w,i} \mid K_{w,i}) \right)^{C_\mathcal{D}(w)} \tag{2.24}$$

met $\Theta(w) = [t_{w,1}, t_{w,2}, \ldots]$ de tokenisatie van woord w door tokeniser Θ. Er zijn vijf mogelijke categorieën: woordgrens (K_0 en $K_{|\Theta(w)|+1}$), prefix, stam, suffix, en "afval", waarover meer in § 4.10. De exacte details van de MLE's van bovenstaande kansen laat ik achterwege, maar weet dat $P(\text{stam} \mid t)$ stijgt met $|t|$ en dat $P(\text{prefix} \mid t)$ en $P(\text{suffix} \mid t)$ stijgen wanneer er veel verschillende types na resp. voor t voorkomen in het corpus, vergelijkbaar met AV in §2.5.2.1. In tegenstelling tot BPE is er geen nood aan extra types waarin een SOW of EOW zit om een prefix of suffix voor te stellen, daar elk type een mix van categoriekansen heeft en dus een andere klasse kan hebben in andere contexten.

Bovendien nestelen Creutz en Lagus (2005a) nu Algoritme 2.4 in een grotere procedure. Eerst trainen ze Morfessor Baseline om een eerste vocabularium en segmentatie

Algoritme 2.4 Pseudocode Morfessor

```
function LEERMORFESSOR(𝒟)
    T ← lege boom
    for w ∈ SPLITSINWOORDEN(𝒟) do
        T ← HERSPLITS(T, w)
    return BLADEREN(T)

function HERSPLITS(T, w)
    c ← AANTAL(T, w)
    (T, ℒ) ← VERWIJDER(T, ℒ, w, c)
    (T_0, ℒ_0) ← VOEGTOE(T, ℒ, w, c + 1, ∅)
    (T*, ℒ*) ← (T_0, ℒ_0)
    (w*_L, w*_R) ← (ε, w)
    for w_L, w_R ∈ TWEESPLITSINGEN(w) do
        (T, ℒ) ← VOEGTOE(T_0, ℒ_0, w_L, c + 1, w)
        (T, ℒ) ← VOEGTOE(T, ℒ, w_R, c + 1, w)
        if ℒ > ℒ* then
            (T*, ℒ*) ← (T, ℒ)
            w*_L, w*_R ← w_L, w_R
    if w*_L ≠ ε then
        T* ← HERSPLITS(T*, w*_L)
        T* ← HERSPLITS(T*, w*_R)
    return T*
```

van het corpus te leveren, waarna ze a.d.h.v. Vgl. 2.24 iteratief proberen om enerzijds subwoordtypes uit V te verwijderen en te vervangen door twee kleinere subwoordtypes, en anderzijds om subwoordtypes toe te voegen met merges zoals BPE. Aangezien ze geen pseudocode publiceren, kan ik daar niet verder op ingaan; de finale tokeniser heet *Morfessor Categories-MAP* (ook wel "Cat-MAP").

Morfessor FlatCat Morfessor Cat-MAP heeft ook een lichtjes andere prior dan Morfessor Baseline: hij betrekt bij de kans van een subwoord niet alleen diens karakters, maar ook de subwoorden die erin vervat zitten. Echter, subwoordkansen zijn van het corpus afhankelijk, dus heeft de likelihood in Cat-MAP invloed op de prior. Grönroos e.a. (2014) vinden die keuze onaanvaardbaar: ze gaat in tegen het hoofdidee achter MAP-modellen, nl. dat de prior en likelihood tegenstrijdig zijn, en hun balans het optimum aangeeft. Ze verwijderen de recursie uit Cat-MAP, waardoor het vocabularium "plat" wordt. Het resultaat, *Morfessor FlatCat*, is minder krachtig dan Cat-MAP gegeven dezelfde trainingdata, maar laat toe om bijkomende informatie uit (platte) referentiesegmentaties te leren. Dat kan Cat-MAP niet, en met die extra data is FlatCat beter.

Morfessor EM+Prune Tot slot bedachten Grönroos e.a. (2020) recent *Morfessor EM+Prune*. Die tokeniser baseert zich op ULM-training (Algoritme 2.3) met twee verschillen: enerzijds verfijnen de auteurs de EM-implementatie die Kudo (2018) vergat te vermelden, en anderzijds maken ze van ULM een MAP-tokeniser door de prior van Morfessor Baseline aan het ML-objectief van ULM toe te voegen. Segmentaties van die combinatie komen beter overeen met morfemen dan die van ULM en die van Morfessor Baseline (zie §4.13 voor meer details over hoe men die overeenkomst meet).

Morfessor-derivaten Er zijn ook enkele tokenisers die een Morfessor-model incorporeren. Banerjee en Bhattacharyya (2018) stellen voor om een corpus eerst te splitsen met Morfessor FlatCat opdat er spaties rond morfeem-achtige strings ontstaan, en die dan met BPE te verwerken om ⟨UNK⟩'s te vermijden (de oorspronkelijke motivatie van BPE, cfr. § 2.1.2). Ze noemen die tokeniser *M-BPE*. Hun hypothese is dat de successen van BPE voornamelijk aan transliteratie te wijten zijn, aangezien BPE zich op frequente substrings (het medium) i.p.v. morfemen (de betekenis) richt. Ze tonen dat een bahdanaumodel met BPE-tokeniser het alleen beter doet dan met Morfessor-tokeniser voor gerelateerde talen (in hun geval Bengaals naar Hindi), en dat eens het model niet meer kan valsspelen (bv. tussen Engels en Hindi) de modellen met Morfessor beter leren. M-BPE is minstens even goed als Morfessor, en soms beter.

Ataman e.a. (2017) passen Morfessor FlatCat aan door $P(\Theta)$ te vervangen door $P(\Theta)^{|V_0|/\tau}$ en noemen dat "*linguistically motivated vocabulary reduction (LMVR)*".[43] Daar is nood aan omdat de prior van Morfessor geen voorkeur heeft voor bepaalde groottes $|V|$ (door de uniforme kans), en dus is het in tegenstelling tot BPE of ULM moeilijk om $|V|$ zelf te kiezen. In een opvolgpaper (Ataman en Federico, 2018) trainen ze bahdanausystemen om verschillende taalfamilies van en naar het Engels te vertalen, waarbij ze Engels met BPE segmenteren en in de andere talen telkens BPE met LMVR vergelijken (met dezelfde τ). Des te morfologisch rijker de taal is waarop LMVR – dus eigenlijk Morfessor – wordt toegepast (in het bijzonder Tsjechisch en Turks), des te meer LMVR uitblinkt boven BPE. In de resterende gevallen wordt alsnog hoger gescoord door een systeem dat beide talen met LMVR segmenteert.

Er is dus kennelijk een verband tussen Morfessor-tokenisatie en morfologie. Mijns inziens is dat om dezelfde reden als waarom ULM beter dan BPE presteert: stel dat we V beperken tot slechts 100 subwoordtypes – karakters niet meegerekend – dan zoekt BPE in feite de 100 strings die het grootste *aantal* hebben, terwijl ULM en Morfessor door hun ML-/MAP-objectief zoeken naar de 100 strings die het meeste *belang* hebben in het corpus. Bovendien werkt BPE bottom-up, en kan het dus pas grotere subwoorden zien eens het die kan vormen met twee reeds besliste subwoorden (zie §4.10).

2.5.3 Bedoeld voor NMT

Er zijn enkele tokenisers die specifiek ontworpen zijn om het *doel*corpus van een NMT-model te segmenteren. In §2.5.1.2 zagen we reeds dat dergelijke tokenisers geen nut meer hebben eens het trainingcorpus gesegmenteerd is, en we ze dus kunnen weggooien nog voor we het NMT-model begonnen is met trainen; immers, tijdens inferentie krijgt het NMT-model alleen een zin in de brontaal, en produceert het token-per-token een zin in de doeltaal, hetgeen impliciet een tokenisatie vormt. De rol van een doelcorpustokeniser is om de doeltaal zo helder mogelijk voor te leggen aan het model, opdat het de tokeniser leert nabootsen en aldus even helder vertaalt.

Het is redelijk om te verwachten dat een TM tijdens inferentie taalfouten aan de invoer te zien krijgt, hetgeen een extra uitdaging is voor de tokeniser van de brontaal. Een voordeel van de onderstaande methodes is dat ze alleen op de doeltaal en dus alleen tijdens training werken, waardoor ze gebruik kunnen maken van modules die enkel op foutloze taal werken.

CSCS-BPE In §2.4.1 beschreef ik reeds kort het cascadesysteem van Huck e.a. (2017) dat ze op Duits toepassen ter voorbereiding van een Engels→Duits-vertaler. Ze recy-

[43] Noteer dat Vgl. 2.11 dan sterk lijkt op een geregulariseerde verliesfunctie in machine learning. Neem immers de logaritme en laat $\lambda = |V_0|/\tau$, dan krijgen we $\ln P(\mathcal{D} \mid \Theta) + \lambda \ln P(\Theta)$.

cleren een bestaande Duitse *stemmer* – een algoritme dat een woord van rechts naar links trimt tot het geen suffices meer herkent uit een vooraf gedefinieerde lijst – om zowel suffix- als prefixgrenzen te vinden, en zetten daarachter de splitser van Koehn en Knight (2003). Elk van die drie zet spaties op de gevonden grenzen, waardoor de BPE-leerfase die daarop volgt geen morfemen kan samenvoegen die gescheiden moeten blijven. Hetzelfde idee zagen we hierboven reeds voor M-BPE; waar de bedoeling van Sennrich e.a. (2016) juist was dat BPE morfemen zou ontdekken, wordt BPE in beide gevallen eigenlijk enkel gebruikt om eender welke woordstam te kunnen coderen met een eindig vocabularium, en niet om nuttige patronen te ontdekken.

Een belangrijk aspect van de cascade is hoe de componenten signaleren waar er oorspronkelijk spaties stonden, gezien ze er valse toevoegen. BPE doet dat klassiek door een EOW-karakter toe te voegen aan een subwoord (cfr. § 2.1.2). Huck e.a. draaien het juist om: a priori nemen ze aan dat er rond *alle* tokens een spatie verschijnt, tenzij naast zo'n lijmkarakter. Prefices krijgen een EOW (`§§`), suffices krijgen een SOW (`$$`): *voorzichtigheid* wordt zo *voor§§ zicht $$igheid*. De drie andere signalisaties zijn innovatief en erop gericht om het vocabularium te minimaliseren: samenstellingssplitsingen en BPE-splitsingen geven de auteurs aan met een *los* token (`@@` resp. `##`), en gelijkaardig voor hoofdletters (`#U`). Stel dat het woord *aaibaar* niet gekend is door de BPE-tokeniser, dan wordt *aaibaarheidsfactor* aan het begin van een zin gesplitst als

`#U` aai `##` baar `$$`heid `@s@` factor

door de cascade, waarin de samenstellingssplitser de interfix inderdaad herkend heeft. De `##`-notatie is gebaseerd op BERT's WPM (zie Tabel 4.1), waarin de bovenstaande tokenisatie in het beste geval eruitziet als

Aai `##`baar `##`heid `##`s `##`factor

waartoe een string dus tot *viermaal* als type kan voorkomen in V: met of zonder hoofdletter, en met of zonder `##`. Huck e.a. motiveren hun losse `#U`, `@@` en `##` door erop te wijzen dat er anders een verspilling van de capaciteit in V is, dat hetzelfde woord met dezelfde betekenis anders op vier verschillende manieren gezien wordt door de decoder (bv. naargelang een woord het begin of het einde van een samenstelling is), en dat de decoder anders niet zomaar elke samenstelling kan vormen.[44]

Huck e.a. evalueren hun Duitse tokeniser in een bahdanau-architectuur.[45] Ze merken dat de decoder leert om hun signalisatie uitbundig te gebruiken (zie hun tabel 7): van alle unieke woorden die hij produceert, bouwt hij 64% m.b.v. een `$$`-suffix en 20% m.b.v. een `@@`-samenstellingstoken. Opvallend: met alleen BPE ($|V| = 50$k) bestaat slechts 13% van alle unieke geproduceerde woorden uit meer dan één token, wat aangeeft dat het merendeel van die 64% gesuffixeerde woorden normaliter gememoriseerd zit in V.

Tot slot merken Huck e.a. dat wanneer ze prefices laten afsplitsen, de BLEU verlaagt. De reden is dat prefices in het Duits net als in het Nederlands een niet-compositionele verandering aan de betekenis geven: de prefix *ver-* zorgt soms voor een connotatie van beweging (*lossen* wordt *verlossen*, *zetten* wordt *verzetten*, *kopen* wordt *verkopen* ...),

[44] Stel dat het vocabularium bv. de types {*analyse*, *factor*, *##factor*} bevat, dan kan de decoder de samenstelling *analyse|factor* met twee tokens vormen, maar *factor|analyse* kan alleen met meer tokens bij gebrek aan het type *##analyse*. De decoder moet dus niet alleen de intentie van de vertaling leren, maar ook wanneer hij die intentie moet opbouwen uit kleine, generieke tokens waarvan hij de karakters niet eens kan zien (cfr. Figuur 1.1).

[45] Voor zover ik weet, heeft er nog niemand dezelfde tokeniser in een transformer getest. Het verschil zou groot kunnen zijn: een bahdanau-architectuur voert alleen attention over tokens in de brontaal, niet in de doeltaal. De besproken signalisatietokens worden geproduceerd in de doeltaal. Een transformerdecoder kan bv. expliciet een samenstellend token `@@` in rekening brengen.

maar meestal is het zelfs een prefix bij een betekenisloze stam zonder die prefix (*verslinden*, *verspillen*, *verliezen*, *vergoeden* ...). Prefices van plaats zijn vaak wel compositioneel (*opkijken*, *aangeven* ...), maar ook niet altijd (*opeten*, *aanranden* ...). De decoder heeft m.a.w. moeite met in te schatten wat voor betekenisverandering het gebruik van zo'n non-compositioneel morfeem heeft, wat opnieuw een argument is voor het memoriseren van de eindige verzameling lexicalisaties in de taal (cfr. §2.4). Ze raden vandaar af om prefices op voorhand af te splitsen. Hun finaal systeem heeft geen naam; Zhou (2018) noemt het "Morph", maar daar ze zelf spreken van een "*c*ascaded *s*uffix + *c*ompound *s*plitting + BPE"-segmentatie, noem ik het *CSCS-BPE*.

MorphGen Tamchyna e.a. (2017) gaan er niet mee akkoord dat de decoder moet leren om de tokeniser van het doelcorpus na te bootsen: enerzijds omdat die tokeniser in het geval van BPE altijd wel ergens tekortschiet en zo de doeltaal onherroepelijk vertroebelt (zie met name §1.4 en §4.11), en anderzijds omdat een autoregressieve decoder inherent woorden genereert door tokens van links naar rechts aaneen te lijmen en dus moeite heeft met niet-concatenatieve woordvorming (zie §4.4).

In plaats daarvan zetten ze het doelcorpus om naar een opeenvolging van lemmata en inflectionele analyses,[46] waardoor het volstaat dat de decoder de betekenis van lemmata en de grammaticale regels van de taal leert, zonder complexe morfologie door de lens van een imperfecte tokeniser. Dergelijke woordvorming kan een volledig deterministische postprocessor achteraf uitvoeren op basis van het lemma en de inflectie. Net als Huck e.a. (2017) zijn ze genoodzaakt om BPE als laatste stap toe te passen op hun doelcorpus, opdat de decoder niet oneindig veel lemmata moet onthouden. De volledige aanpak noemen ze *MorphGen*.

DPE CSCS-BPE en MorphGen verwerken beide de doeltaal van een parallel corpus, maar behandelen het alsof het een monolinguaal corpus is. Ze halen geen informatie uit de parallelle brontaalzinnen die we ter beschikking hebben, terwijl een vertaling juist bijzonder nuttig kan zijn ter desambiguatie van woorden en hun segmentaties, bv. in talen zonder spaties (Chinees, vandaar dat CWS nodig is) of met non-concatenatieve woordvorming (Hebreeuws, zie Klein en Tsarfaty (2020) en §4.4).

Op basis van die ideeën stellen He e.a. (2020) een tokeniser voor die voor elke doeltaalzin de meest waarschijnlijke segmentatie kiest uit alle mogelijke segmentaties, en bij die beslissing gebruik maakt van de brontaalzin. Net als Kudo (2018) zien ze de segmentatie van een zin met karakters $y_1 \dots y_n$ als een toevalsvariabele met 2^{n-1} uitkomsten, waarover ze de uitkomst met maximale kans willen bepalen; in tegenstelling tot Morfessor (Creutz en Lagus, 2002), ULM (Kudo, 2018) en S-BPE (Vilar en Federico, 2021) verwerpen ze echter de unigram-aanname dat tokens onafhankelijk van hun omringende tokens voorkomen, om een exact model te krijgen. Dat is problematisch: de unigram-aanname zorgde ervoor dat al die tokenisers de maximale kans konden vinden in $O(n^2)$ met een viterbialgoritme. Door een tokensequentie $t_1, \dots, t_m$ te modelleren met de kettingregel

$$P(t_1, \dots, t_m) = \prod_{i=1}^{m} P(t_i \mid t_1, \dots, t_{i-1}) \neq \prod_{i=1}^{m} P(t_i) \tag{2.25}$$

is er geen binnenweg mogelijk met een viterbialgoritme en is de kost $O(2^n)$. De truc die

[46] Voor een werkwoord is dat een tupel (vorm, wijs, tijd, persoon, aantal). Voor een naamwoord is dat een tupel (naamval, genus, getal).

ze voorstellen is om niet te conditioneren op de voorgaande *tokens*, maar *karakters*:

$$P(t_1, \ldots, t_m) \equiv \prod_{i=1}^{m} P(t_i \mid \text{concat}(t_1, \ldots, t_{i-1})). \tag{2.26}$$

In §B.2 toon ik hoe en waarom dat wel te maximaliseren is in $O(n^2)$ tijd. Dat is op zich geen nieuw idee: Grave e.a. (2019) beschrijven exact het causale taalmodel van Vgl. 2.26. Er resteren twee zaken te bepalen: wat voor model we gebruiken om de kans te krijgen van een token gegeven de voorgaande karakters, en het vocabularium waarvan die tokens afkomstig zijn.

Grave e.a. kiezen als vocabularium alle karakterbigrammen, d.w.z. $V = \{$aa, ab, ac, ..., ba, bb, bc, ...$\}$. He e.a. gaan ervan uit dat V gegeven is, zonder er verder op in te gaan. In beide gevallen is de method indifferent voor hoe V verkregen is: vocabularisatie en inferisatie zijn van elkaar gescheiden (cfr. §2.5.1.2).

Stel dat we een trainbaar model hebben om een kans $P(Y_i \mid y_1, \ldots, y_{i-1})$ mee te voorspellen, met $y_1 \ldots y_{i-1}$ karakters en met $Y_i \in V$ een token bestaande uit één of meer karakters. We kunnen zo'n model niet gesuperviseerd trainen: dat zou immers vereisen dat we Y_i reeds kenden, en dus wisten wat de tokenisatie van de invoer was, terwijl we juist de tokeniser aan het bouwen zijn. Vandaar moet de verliesfunctie per trainingzin $y_1 \ldots y_n$ sommeren over alle mogelijke tokensequenties die die zin vormen bij concatenatie – m.a.w. *marginaliseren* over de segmentatievariabele. Dat gebeurt efficiënt met hetzelfde viterbialgoritme om het maximum te berekenen, alleen is het maximum een som geworden. Het idee van marginaliseren over segmentaties bij gebrek aan gekende tokenisatie komt ook voor in het *segmenteel LM (SLM)* van Sun en Deng (2018), een volledig karaktergebaseerd CLM (in invoer én uitvoer) dat in staat is om segmentatiegrenzen aan te duiden, en zo bruikbaar is voor CWS.

Downey e.a. (2022) verrijken SLM's – die causaal zijn, want viterbirecursie werkt alleen unidirectioneel – met een beperkte hoeveelheid extra informatie van een bidirectioneel MLM. Gezien He e.a. hun extra informatie uit een andere taal halen en dus niet de te segmenteren zin, stellen zij een *mixed character-subword (MCS) transformer* voor als kansmodel: de encoder heeft zoals bij Vaswani e.a. (2017) brontaalsubwoorden $X_1, \ldots, X_L$ als invoer en produceert er subwoordembeddings voor, maar de decoder heeft nu karakters $y_1, \ldots, y_i$ als autoregressieve invoer, zelfs al voorspelt de decoder nog steeds het volgende subwoord Y_{i+1} en niet het volgende karakter. Zo krijgen we de kans $P(Y_{i+1} \mid y_1, \ldots, y_i, X_1, \ldots, X_L)$ om te gebruiken in Vgl. 2.26. Omdat een viterbialgoritme een voorbeeld is van *dynamic programming* en de decoder er zowel een gebruikt tijdens het trainen als tijdens corpussegmentatie,[47] noemen He e.a. (2020) hun tokeniser *dynamic programming encoding (DPE)*.

DPE segmenteert een woord in de doeltaal anders naargelang de karakters die ervoor komen en de vertaling in de brontaal, wat sterk verschilt van de "one-size-fits-all"-aanpak van BPE (zie §4.17). De auteurs ondervinden o.a. dat DPE >50% van woorden met frequentie ≤50 anders segmenteert dan BPE. Tegelijk merken ze ook dat DPE nog beter werkt indien de brontaal met BPE-dropout wordt gesegmenteerd.

[47] Let op: de MCS-transformer is niet het NMT-model, maar de tokeniser. Een groot verschil is bijvoorbeeld dat in NMT de decoder op voorhand niet weet wat de uitvoerzin is, en dus met een greedy/beam search zelf een benaderend voorstel moet doen. De MCS-transformer, daarentegen, krijgt de uitvoerzin reeds gegeven, en dient alleen als het kansmodel waarvan de viterbisegmenteerder uit §B.2 gebruik maakt om Vgl. 2.26 te maximaliseren. Eens de MCS-transformer het doelcorpus gesegmenteerd heeft, mogen we hem weggooien en wordt het daadwerkelijke NMT-model getraind – toevallig ook een transformer voor He e.a. (2020), alleen dit keer van subwoorden naar subwoorden.

2.5.4 Zonder subwoorden

Er is een groeiende stroming die de concepten van "tokeniser", "vocabularium" en "subwoordtype" volledig verwerpt, met als hoofdargument dat tokenisers zoals hierboven beschreven bevroren worden eens het model begint met trainen. Dat laatste gebeurt in neurale modellen immers door continue modelparameters een klein duwtje te geven tegen de gradiënt van de verliesfunctie in (*gradient descent*), en dat is niet compatibel met de niet-neurale en discrete aard van bv. de keuze van vocabularium. In plaats daarvan is het voorstel om modellen op embeddings van karakters (of bytes) toe te passen, en de aggregatie tot grotere eenheden over te laten aan een trainbare module deel van de rest van de architectuur. Dat alles ondanks het afschrijven van karaktermodellen als "suboptimaal" door Sennrich e.a. (2016).

ByT5 In § 2.2.2 zagen we T5, een transformer die als taalmodel gebruikt wordt. Xue e.a. (2022) passen T5 aan door een vocabularium van enkelvoudige bytes te gebruiken, en de transformerencoder drie keer zoveel lagen als de decoder te geven. Het resultaat, *ByT5*, werkt even goed als T5 en is robuuster tegen typo's.

CharacterBERT Een model dat geen subwoorden gebruikt, kan nog steeds nut halen uit spaties die woordgrenzen afbakenen. Vandaar stellen El Boukkouri e.a. (2020) een variant van BERT voor die embeddings verwerkt op het woordniveau i.p.v. het subwoordniveau zoals Devlin e.a. (2019). Daartoe splitsen de auteurs eerst elk woord in karakters,[48] en gebruiken ze een triviale embedding-LUT om die naar 16-dimensionale vectoren om te zetten. Vervolgens contextualiseren ze de rij embeddings van elk woord door er meerdere convoluties in parallel op uit te voeren (telkens met een *convolutioneel neuraal netwerk (CNN)*), waarvan de uitvoer geconcateneerd wordt en o.i.v. een *snelwegnetwerk* (Eng.: *highway network, HN*) complexe transformaties ondergaat die eindigen bij één 768-dimensionale embedding per woord, het invoerformaat van BERT.

De aaneenschakeling van de karaktermodule en BERT noemen ze *CharacterBERT*. Tijdens het trainen propageren de gradiënten nu tot voorbij de invoer van BERT naar de CNN's en het HN, waarbij het combineren van karakters met continue stapjes verbetert. Dat kan niet in een subwoordtokeniser, wat o.a. gevolgen heeft op het leren van bijkomende woordenschat tijdens het finetunen (zie §4.9).

CANINE Niet alle talen hebben spaties, maar logischerwijs staan de karakters van één woord altijd naast elkaar. Clark e.a. (2022) stellen net als CharacterBERT een omvormer van karakterembeddings naar grotere embeddings voor, alleen zonder rekening te houden met spaties en alleen met nabijheid. Eerst contextualiseren ze de embeddings met één transformerencoderlaag die bovendien slechts attention kan voeren in een omgeving van 128 karakters. Vervolgens voeren ze een convolutie uit (met leerbare impulsrespons) die blokken van 4 karakters omzet in één embedding, en die "pseudo-subwoorden" geven ze aan BERT.

Door het downsamplen in blokken van 4 karakters is er geen directe link tussen de BERT-embeddings en de invoer. Dat lossen ze op door elke uitvoer van BERT viermaal te kopiëren, te concateneren met de embeddings vóór de convolutie, en dan de omgekeerde operatie toe te passen (een convolutie en een laatste transformerlaag). Het geheel heet de *character architecture with no tokenization in neural encoders (CANINE)*.

[48] In de praktijk werken ze op bytes zoals BBPE, ondanks de naam van het model. Er zijn immers heel veel karakters, maar slechts 256 bytes (zie §4.12).

Charformer De bovenstaande drie krijgen kritiek van Tay e.a. (2022): ByT5 is computationeel te zwaar, CharacterBERT werkt alleen in talen die woorden met spaties scheiden, en CANINE's attention en convoluties zijn te lokaal en hebben een ingebakken blokgrootte. Dat laatste kaarten de auteurs aan met de *Charformer*: ze beginnen met karakterembeddings die ze elkaar laten informeren met een convolutie. Vervolgens groeperen ze niet alleen opeenvolgende blokken van 4 karakters zoals CANINE, maar ook van 3, 2 en 1 karakter in parallelle kopieën.[49] Elke blok wordt samengevat als het gemiddelde van zijn embeddings.[50] Een dense laag zet elke blokembedding om naar een reële score, waarna elk karakter een softmax neemt over de scores van de blokken waarin het zich bevindt.

Op dat moment heeft karakter i een rij blokembeddings $\vec{B}_{1,i} \dots \vec{B}_{4,i}$ en een rij gewichten $p_{1,i} \dots p_{4,i}$. Nadat er een laatste keer self-attention (§ 2.2.2) gevoerd is overheen de gewichtsvectoren (opdat de karakters gewichten kunnen afkijken van elkaar), nemen Tay e.a. de gewogen som van de gewichten en blokken per karakter. Van de resulterende rij embeddings nemen ze het gemiddelde in groepen van 2 bij wijze van downsampling, en die embeddings geven ze aan BERT als "latente subwoorden".

Het bovenstaande proces noemen ze *gradient-based subword tokenisation (GBST)*. Een visualisatie van de gewichtvectoren van elk karakter toont o.a. dat GBST in het Engels automatisch leert om hoge scores toe te wijzen aan klinkers en spaties in de blokken van grootte 1.

PIXEL In Hoofdstuk 1 verkende ik hoe een tokeniser dienst doet als de leesbril van een NLP-model. Rust e.a. (2023) maken de analogie nog explicieter met de *pixel-based encoder of language (PIXEL)*: ze zetten tekst eerst om naar een screenshot, en geven die dan aan de *visuele transformer (ViT)* van Dosovitskiy e.a. (2021) die de tekst letterlijk op zicht verwerkt. Een ViT splitst een afbeelding op in vierkante pixelgebieden, laat die elk verwerken door een dense laag (een dynamisch alternatief op de embedding-LUT), en behandelt het resultaat zoals een taaltransformer.

Tekst met een visueel model verwerken is op eerste zicht nogal omstreden, maar het heeft wel degelijk meerwaarde: Unicode is een gediscretiseerde versie van schriften, terwijl veel schriften hun betekenis overbrengen a.d.h.v. het aanbrengen van handgeschreven details die geen eindig aantal posities hebben (bv. lijntjes in Chinese tekens). Aparte embeddings toewijzen aan alle Unicode-codepunten van dergelijke schriften zorgt voor redundantie, die verdwijnt door de tekst visueel te interpreteren.

§4.12.4 bespreekt enkele nadelen van grote vocabularia, waaronder een grote embedding-LUT. In de papers van de vijf bovenstaande end-to-end-modellen komt het telkens aan bod dat men een significant deel van het parameterbudget van de transformer kan terugwinnen door het elimineren van de embedding-LUT (volgens Chung e.a. (2021) tot wel 71%), dat dan bv. herbesteed wordt in een diepere encoder.

[49] Door de convolutie in het begin is het niet zo erg dat sommige aangrenzende karakters nooit in dezelfde blok zitten.

[50] Een positionele codering is niet nodig, opnieuw door de convolutie in het begin.

H3 Probleemstelling

Uit de literatuur omtrent het leren en gebruiken van subwoordtokens schijnt er kennelijk weinig consensus te zijn over de *beste* manier om te tokeniseren. Er is wel een rode draad: BPE wordt gezien als het vertrekpunt (voor varianten) of het referentiepunt (voor alternatieven) van andere tokenisatiealgoritmes. Wanneer tokenisatie in de praktijk wordt toegepast (bv. in transformermodellen en de daarvan afgeleide GPT en RoBERTa), dan heeft BPE bovendien de reputatie van *goed genoeg* te zijn, alsof er geen nadelen aan die ontwerpkeuze verbonden zijn.

Zoals besproken in de vorige twee hoofdstukken was BPE in de eerste plaats een historische doorbraak, waaraan het geen onbelangrijk deel van zijn populariteit te danken heeft. Sennrich e.a. (2016) publiceerden inderdaad het eerste algoritme dat het probleem van het open vocabularium remedieerde door de tekst te splitsen (i.p.v. om te zetten naar een arbitraire nieuwe codering zoals Chitnis en DeNero (2015)) en bevattelijker was dan WordPiece (Schuster en Nakajima, 2012). Op die manier is "tokenisatie in subwoorden" sterk geassocieerd geraakt met "BPE gebruiken".[51]

Dat er überhaupt varianten bestaan, doet al vermoeden dat het schoentje ergens knelt, en de inburgering van BPE als *de facto* standaard betekent niet dat het gebruik van BPE zonder technische schuld (Eng.: *technical debt*) komt. Integendeel: een BPE-tokeniser is een vooraf bevroren statistische component in het tijdperk van neurale transferlearningarchitecturen. Als zo'n rigide tokeniser enige inductieve bias heeft, dan zal het dus moeilijker zijn om die te verzachten, i.t.t. end-to-end-systemen (cfr. § 2.5.4).

3.1 Onderzoeksvragen

Het is bijgevolg oppassen met de keuze van tokeniser. Vandaar onderzoekt deze thesis enerzijds de technische schuld die men aangaat door BPE te gebruiken:

Onderzoeksvraag 1 *Wat zijn de problemen die BPE-tokenisatie teweegbrengt?*

Identificatie van mankementen is behulpzaam, maar er kan pas verandering komen in hoe men beslist te tokeniseren in de toekomst indien er betere tokenisers beschikbaar

[51] Naamherkenning krijgen als gevolg van een innovatie is niet ongewoon: we zien dergelijk pioniersvoordeel (Eng.: *first-mover advantage*) in de Nederlandse woorden *kodak* (camera met filmrol), *bic* (balpen) en *pamper* (luier), afkomstig van de respectievelijke merknamen Kodak, Bic en Pampers die de producten het eerst massaal commercialiseerden. Men spreekt van *genericide*.

zijn. We hebben gelukkig een hoop BPE-varianten en -alternatieven gezien in §2.5, en sommige daarvan werden ontwikkeld met het oplossen van een specifiek probleem in het achterhoofd.

Wat we echter niet zagen, zijn BPE-tokenisers waarin inferisatie (gedefinieerd in §2.5.1.2) meer dan een bijproduct van vocabularisatie is – maar er i.t.t. DPE nog wel aan gekoppeld is – noch BPE-tokenisers met een niet-hiërarchische segmentatieregel.

Onderzoeksvraag 2 *Zijn de problemen met BPE-tokenisatie op te lossen met aanpassingen aan inferisatie en/of segmentatie?*

Noteer dat we niet op zoek gaan naar een alternatieve tokeniser die alle problemen tegelijk oplost. De analyse van Onderzoeksvraag 1 kan (en zal) uitwijzen dat BPE tegenstrijdige eigenschappen heeft, waardoor eenzelfde aanpassing aan BPE nooit elk probleem kan oplossen. Wel zijn de problemen afzonderlijk aan te pakken, of is er een deelverzameling van prioritaire problemen die alsnog tezamen oplosbaar zijn.

3.2 Verder verloop

De rest van de thesis is ingedeeld volgens de onderzoeksvragen: Onderzoeksvraag 1 wordt uitvoerig behandeld in Hoofdstuk 4 a.d.h.v. de literatuur en bijkomende experimenten met Nederlandse datasets, en Hoofdstuk 5 stelt enkele innovaties voor ten behoeve van Onderzoeksvraag 2.

Deel II

Contributie

H4 Analyse van problemen met BPE

We zagen in §2.5 een heleboel varianten op BPE die al dan niet ontworpen waren om een specifiek mankement van het algoritme op te lossen. Algemener valt het op dat er in veel papers waarin BPE-tokenisatie aan bod komt een losse opmerking te vinden is over zo'n gebrekkigheid; desalniettemin is er nog nooit een compilatie van al die individuele kritieken opgesteld, voor zover ik weet. Vandaar bespreekt elke sectie in dit hoofdstuk zo'n kritieke noot. De lijvigheid van de lijst wijst erop dat NLP-onderzoek dringend de status van BPE als *de facto* standaard moet herzien.

Waar toepasselijk eindigt de sectietitel telkens met de citatie van de vroegste paper waarin ik de kerngedachte van die sectie heb teruggevonden. Dat wil niet zeggen dat de auteurs een uitbundige analyse van het probleem hebben gedaan noch het probleem hebben opgelost; soms gaat het over een enkele zin, waaraan ik vervolgens verdieping geef. Daardoor is dit hoofdstuk meer dan een tweede literatuurstudie. Voor ideeën die onvoldoende gestaafd zijn, voer ik verifiërende experimenten uit met de Nederlandse BBPE-tokeniser van RobBERT-2020 (Delobelle e.a., 2020) met $|V| = 40\text{k}$, hierna kortweg BPE_{nl} genoemd, te vinden op HuggingFace. Meer informatie over de gebruikte datasets is te vinden in Appendix C.

4.1 Onvolledig alfabet (Radford e.a., 2019)

> **Samengevat** – Het alfabet van BPE, d.w.z. de tekens waarmee V wordt geïnitialiseerd, is ofwel te klein om andere talen te ondersteunen, ofwel juist veel te groot indien het een brede ondersteuning voor Unicode biedt.

De belofte van Sennrich e.a. (2016) is dat hun BPE elke invoer kan omzetten tot id's zonder ⟨UNK⟩'s in te voegen. Zelfs in het ergste geval waarbij er geen enkele substring van een invoerwoord bekend is, kan de tokeniser terugvallen op de karakters waarmee V is geïnitialiseerd (cfr. lijn 4 in Algoritme 2.1), hetgeen ong. met een *alfabet* overeenkomt.

In de praktijk wordt die belofte niet waargemaakt, omdat corpora uit meer dan alleen ASCII-karakters bestaan, bv. exotische alfabetten (Grieks, cyrillisch, Arabisch, Chinees-Japans-Koreaans (CJK), brahmische schriften ...), en allerlei pictogrammen waaronder emoji op sociale media. Een "karakter" in UTF-8-tekst komt algemeen eerder overeen met een Unicode-codepunt, maar zo zijn er 149 186 (Unicode Consortium, 2022). Gezien er tienduizenden exotische "letters" zijn, is het niet voldoende om alleen de pictogrammen te laten vallen.

Lample en Conneau (2019) geven aan dat XLM een alfabet van 15 000 codepunten heeft. Ofwel is het alfabet dus onvolledig voor meertalige teksten (resulterend in veel ⟨UNK⟩'s), ofwel eist het een groot deel van de totale capaciteit $|V|$ op (cfr. §4.12) nog vóór een enkel BPE-subwoord is gevocabulariseerd. Radford e.a. (2019) observeerden dat probleem voor het eerst bij het trainen van GPT-2, en gaven en passant een oplossing die later BBPE werd gedoopt (cfr. §2.5.2.2):

> Despite its name, reference BPE implementations often operate on Unicode code points and not byte sequences. These implementations would require including the full space of Unicode symbols in order to model all Unicode strings. [...] In contrast, a byte-level version of BPE only requires a base vocabulary of size 256.

De keerzijde van BBPE is dan weer dat extra aandacht moet besteed worden aan het opkuisen van het corpus indien we ons effectief tot één taal willen beperken. Het deel van OSCAR (Suárez e.a., 2019 en §C.1) dat zogezegd alleen Nederlandse tekst bevat (en waarop BPE_{nl} is getraind), bevat 20 812 149 tekstfragmenten waarvan 169 314 met minstens één exotisch karakter. Figuur D.10 toont een verdeling van die karakters op basis van de alfabetten in Tabel C.1. In het slechtste geval zorgen ze voor vreemde byteparen in V; zo zijn er 510 types in BPE_{nl} waarvan de karakters niet tot ASCII of ISO 8859-1 behoren, en dus zeker niet Nederlands zijn. Tabel D.1 toont er enkele. Dat ze ook geen ander alfabet nabootsen, doet vermoeden dat ze afkomstig zijn van byte-artefacten in webtekst (bv. grote spannes van speciale witruimte).

Er bestaat naast BBPE een andere, triviale oplossing op het bovenstaande probleem: waarom niet gewoon de maximumgrens $\tau = |V|$ optrekken om een gigantisch initieel alfabet toe te laten? Dat lijkt voor de hand te liggen, maar heeft te veel sterke nadelen, zie opnieuw §4.12. Een mildere versie daarvan is de "character coverage"-parameter in de BPE-implementatie van het SentencePiece-pakket, die controleert welke fractie van het corpusalfabet (geordend op frequentie) in V wordt opgenomen.

Tokenloze tokenisers (cfr. §2.5.4) gaan de andere kant op: ze hebben wel een bytevocabularium, maar geen grotere subwoorden. Dat vermijdt het opnemen van rare types zoals hierboven. We zagen ook reeds dat PIXEL een goed idee kan zijn voor grote alfabetten.

4.2 Cijfers als letters (Rogers e.a., 2021)

> **Samengevat** – BPE behandelt cijfers alsof ze letters zijn, waardoor er meercijferige getallen in het subwoordvocabularium zitten.

Tokenisatie is bedoeld voor tekst met voorspelbare structuur. Getallen zijn ongestructureerd[52] en komen desalniettemin veel voor in lopende tekst. Daardoor bevat BPE_{nl} bv. 564 types die uit cijfers bestaan, waarvan 374 (2 op 3) met SOW (zie § 2.1.2 en §4.3 voor een verdere bespreking van SOW en EOW).

Bostrom en Durrett (2020) tonen als voorbeeld dat BPE het getal 1848 als *_184|8* splitst. ULM splitst het niet op. Beide tokenisers hebben dus bedenkelijke subwoorden in V: de vraag is inderdaad of het überhaupt nuttig is om getallen te tokeniseren met subwoorden groter dan één cijfer. Mijns inziens is het dat niet, want het doel van subwoordtokenisatie is (cfr. Hoofdstuk 1) om meer invoer dan gekende woorden te kunnen voorstellen, maar ook om betekenisvolle statische embeddings toe te laten voor grotere eenheden dan karakters. Getallen hebben daar niets aan. Rogers e.a. (2021) menen zelfs dat getallen in grotere eenheden segmenteren schadelijk is voor het taalbegrip van BERT, gezien twee numeriek nabije getallen er compleet anders kunnen uitzien; BPE_{nl} segmenteert 369 als

[52] Cijfers zijn in de praktijk onderhevig aan bv. de wet van Benford, maar geen morfologie.

_36|9 maar 370 als _3|70, en dat effect verergert voor grotere getallen (bv. _150|7|24|4 versus _150|72|45)

Cijferembeddings dienen hoogstens voor transliteratie. Bubeck e.a. (2023) tonen in de "*Sparks of Artificial General Intelligence*"-paper dat GPT-4 kan "leren" (of beter, geïnstrueerd worden met context) om de API van een rekenmachine op te roepen (zie hun figuur 5.2). Daartoe moet GPT-4 dus de cijfers van een getal uit de invoertekst identiek kunnen reproduceren in de uitvoertekst.

4.3 Spatie in subwoordtypes

> **Samengevat** – Er is geen consensus over de voorstelling van spaties in subwoordtokenisatie.

De string aan de invoer van de tokeniser bestaat uit karakters die een zin voorstellen, dus de invoer bevat spaties. Een BPE-tokeniser verwijdert die spaties en segmenteert elk woord apart, waarna het de tokens concateneert tot één grote sequentie. Uit die sequentie moet afleidbaar zijn of de grens tussen twee tokens een spatie nodig heeft of niet, want anders ontstaat er ambiguïteit, en informatieverlies t.o.v. de invoer.

Er zijn drie gangbare manieren om spaties aan te geven in een tokensequentie. De eerste is die van Schuster en Nakajima (2012), nl. om kopieën van het alfabet aan het alfabet toe te voegen, waarin er aan de types een spatiekarakter is geplakt. WPM neemt drie kopieën (cfr. het voorbeeld in §2.5.2.1): de ene krijgt telkens een SOW (_) vooraan, de ander een EOW (ook _) achteraan, en de laatste krijgt beide. Sennrich e.a. (2016) doen het anders: na het splitsen in woorden doen ze alsof er een extra, zelfstandig EOW-karakter (`</w>`) achter elk woord hangt, dat ze aan het alfabet hebben toegevoegd en al dan niet met de voorgaande letter(s) merget. CSCS-BPE van Huck e.a. (2017) hanteert een derde stijl, nl. om gewoonweg een spatietoken te voorzien.

Voor de volledigheid vermeld ik dat de implementatie van bovenstaande bronnen in de praktijk durft verschillen van hun paper. Zo maakt WPM volgens Devlin e.a. (2019) en HuggingFace slechts één kopie van het alfabet, en krijgen bovendien alle subwoordtypes búíten die aan het begin van een woord een SOW (`##`). Dat is equivalent met een SOW aan het begin van een woord, aangezien het model de onderliggende tekst niet kan zien en alleen weet dat er twee soorten types zijn. Sennrich e.a. hebben hun implementatie op GitHub dan weer aangepast om nu wel een kopie te nemen, nog steeds met EOW (`</w>`). Desniettegenstaande trainden Delobelle e.a. (2020) BPE_{nl} met de oude versie van BPE, hoewel dit keer met een zelfstandige SOW (`Ġ`) i.p.v. een zelfstandige EOW.

Voor zover ik weet, is er geen consensus over welke variant beter is. Er zijn naast de stijl van CSCS-BPE in feite vier combinaties mogelijk, samengevat in Tabel 4.1. Tokenisers in dezelfde kolom kunnen in principe op hetzelfde vocabularium uitkomen. Die in de eerste rij verergeren de situatie van §4.1 en §4.12. Anderzijds is er zoals gewoonlijk in informatietheorie een trade-off tussen de lengte van de codering (aantal tokens nodig voor de invoer) en de overhead van het codeboek (het vocabularium): ceteris paribus zorgt een apart spatietoken bv. voor een verdubbeling van de invoerlengte, maar hebben alle types exact één variant.

Hebben SOW en EOW nut? Mogelijks: een SOW kan vroegtijdig stoppen met het mergen van de letters van een prefix indien die in het midden van een woord voorkomen, op voorwaarde dat de mergeboom gebalanceerd is (cfr. §4.10) en de SOW niet pas als laatste wordt toegevoegd. Analoog voor EOW en suffices. Anderzijds kunnen prefices en suffices wel degelijk zonder spatie voorkomen, nl. in samenstellingen (bv. *beleggingsverzekering*).

	met SOW	met EOW
gedwongen	¬BERT-WPM	BPE v0.2
geleerd	BPE_{nl}	BPE v0.1

Tabel 4.1 – Mogelijke manieren waarop spaties in subwoordtypes kunnen belanden. "¬BERT-WPM " is het complement (en equivalent) van hoe Devlin e.a. (2019) WPM aangeven, bv. `##tok|en|iser` i.p.v. `tok|##en|##iser`.

In § 4.16.3 zal blijken dat de keuze tussen SOW en EOW niet triviaal is voor samenstellingen, en BPE_{nl} daartoe een nadelige verdeling heeft.

Welk van beide is beter? Mijns inziens zijn ze beide een slappe benadering van het driecategoriesysteem (prefix, stam, suffix) van Morfessor Cat-MAP (Creutz en Lagus, 2005a), dat net als CSCS-BPE geen spaties in types heeft, en met een HMM (Vgl. 2.24) *per woord* probabilistisch beslist tot welke categorie een token behoort.

4.4 Concatenatief (Amrhein en Sennrich, 2021)

> **Samengevat** – BPE modelleert alleen woordvorming d.m.v. samenplaatsing van letters. Interactie of opbreking van letters zorgt voor performantieverlies.

In Hoofdstuk 1 werd subwoordtokenisatie gemotiveerd op basis van verschillende Europese talen. De woordvorming van dergelijke talen is voornamelijk *concatenatief*, d.w.z. dat morfemen gewoon aan elkaar geregen worden, en het dus steek houdt om woorden te analyseren als groepjes (tokens) van opeenvolgende letters. Andere talen hebben complexere woordvormingsregels: in het Hebreeuws (Klein en Tsarfaty, 2020) gebeurt inflectie bv. vaak met een *infix*, i.e. een partikel dat men middenin een woordstam plaatst ter verbuiging. Dat zorgt ervoor dat de woordstam niet meer met hetzelfde subwoord te tokeniseren is, gezien de letters niet meer opeenvolgend zijn. Bovendien zijn Hebreeuwse morfen (de verschijningsvormen van morfemen) vaak veel kleiner dan de morfemen zelf (bv. één letter), waardoor een string meerdere geldige segmentaties en interpretaties kan hebben, waarbij de juiste alleen uit de (bidirectionele) context volgt.

Amrhein en Sennrich (2021) onderzoeken in welke mate een NMT-transformer met BPE-tokeniser kan omgaan met een aantal non-concatenatieve woordvormingsprocessen. Aangezien er geen enkele taal is waarin alle door hun onderzochte processen voorkomen, en er geen grote corpora bestaan die ze expliciet aanduiden met annotaties, stellen Amrhein en Sennrich regels op die bestaande Duitse en Engelse zinsdelen aanpassen om de onderzochte processen na te bootsen. Zo wordt bv. het voorzetsel *auf* vervangen door het nonsenswoord *siye* en wordt zijn vertaling *on* vervangen door de artificiële infix *dezaxe*, zodat *on my birthday* vervangen wordt door *my bdezaxeirthday*.
Ook modelleren ze *klinkerharmonie*, een non-concatenatief proces in o.a. het Hongaars en Turks waarbij klinkers in hetzelfde woord zich aan elkaar aanpassen opdat ze dezelfde tongpositie hebben: men zegt *Türkiye'de* (Ned.: *in Turkije*) maar *Avrupa'da* (Ned.: *in Europa*). Hongaars en Turks zijn beide agglutinatief, waardoor woorden heel veel morfemen bevatten die door klinkerharmonie allemaal hun klinkers moeten synchroniseren.[53] Amrhein en Sennrich voegen artificiële klinkerharmonie toe door een voorzetsel te vervangen door een adjectief dat altijd dezelfde drie medeklinkers heeft maar de laatste twee klinkers van het bijhorende substantief overneemt: *before yesterday* wordt *yesterday bekam*, maar *before morning* wordt *morning bokim*.

Ze vergelijken vier tokenisers: zuiver karakters, BPE met $|V| = 32\text{k}$, en BPE-dropout

[53] Een verwant effect is de Duitse klankverschuiving d.m.v. een *umlaut*.

met $|V| = 500$ of $|V| = 32\text{k}$. Alle modellen lijken klinkerharmonie te kunnen begrijpen en vertalen naar hetzelfde woord, maar klinkerharmonie reproduceren lukt alleen goed ($> 80\%$ accuraat) voor de twee kleine vocabularia. Het model met BPE is meestal slechter dan het model met BPE-dropout.

Hoewel ze zelf opperen dat het mogelijks gemakkelijker is om met klinkerharmonie om te gaan als het méér in het corpus voorkomt, meen ik dat dat het juist moeilijker maakt voor de tokeniser. Er is dan namelijk een veel grotere variëteit aan subwoorden – terwijl ze eigenlijk gegroepeerd moeten worden in templates zoals *b_k_m* hierboven – waardoor BPE met dezelfde $|V|$ minder morfemen kan vatten. Bovendien moet een NLG-model preciezer kunnen kiezen tussen die gerelateerde subwoorden, t.o.v. een taal zonder klinkerharmonie: niet alleen moeten de latere subwoorden van een woord conformeren in klinkersoort met de eerdere, maar de eerdere subwoorden moeten in feite zo gekozen zijn dat ze de keuze van de latere niet in het gedrang brengen. Dat vereist planning, wat mist in autoregressieve decoders (Bubeck e.a., 2023).

Tay e.a. (2022) merken overigens op dat zelfs tokenloze modellen zoals Charformer het moeilijk hebben met non-concatenativiteit vanwege het gebruik van groeperingen en/of convoluties van nabije karakters.

4.5 Deterministisch (Kudo, 2018)

> **Samengevat** – BPE-tokenisers laten niet toe om meerdere segmentaties van dezelfde zin te samplen.

Kudo (2018), Bostrom en Durrett (2020) en Provilkov e.a. (2020) tonen allen de regulariserende voordelen van het samplen van een probabilistische tokeniser om LM's aan meer dan één segmentatie van dezelfde invoer bloot te stellen (cfr. §2.5.2.3). SLM's trekken dat door tot het rekening houden met álle mogelijke segmentaties. Die regulariserende voordelen heeft BPE niet.

BPE-dropout brengt een wijziging aan (Algoritme 2.2) die BPE probabilistisch maakt door een kansmassa over de verzameling van segmentaties van de invoer te definiëren. Het is echter onduidelijk wat voor informatie er in die kansmassa vervat zit: de kans om één merge te laten vallen is een vaste p onafhankelijk van de merge, dus daar zit geen informatie in. De volgorde die M aan de merges oplegt, geldt wel: als tijdens een zekere iteratie de beste merge met kans p wegvalt, dan wordt de tweede beste merge gekozen als die met kans $(1-p)$ niet wegvalt. Er geldt dus een soort geometrische verdeling overheen de kansen binnen één iteratie. Bovendien verschilt de lijst van merges naargelang de merge die in de vorige iteratie gekozen is, en die lijst wordt door V bepaald. De kans om een zekere lijst te bereiken volgt dus ook een soort geometrische verdeling. Mijns inziens modelleren die kansen niets meer dan het BPE-dropout-proces zelf.

Men zou meer informatie kunnen injecteren door p per merge te laten variëren. Aangezien BPE naast de mergevolgorde ook een metriekwaarde uit het corpus afleidt, zou die een kandidaat kunnen zijn om p te informeren; het probleem is echter dat er geen vanzelfsprekende interpretatie van de BPE-metriek (cfr. §2.5.2.1) als kans bestaat: zo kan bv. de frequentie van eenzelfde paar sterk afnemen doordat een doorgevoerde merge een deel van zijn tokens steelt, waardoor de frequentie onbetrouwbaar is overheen iteraties.

4.6 Contextloos (Klein en Tsarfaty, 2020)

Samengevat – BPE segmenteert woorden in isolatie, zonder omringende context.

We zagen in § 2.5.3 dat He e.a. (2020) de DPE-segmentatie op twee vlakken trachten te conditioneren: de kans op een token in de uitvoer hangt af van de volledige invoer, en ook van alle voorgaande karakters van de uitvoer. Die kans kon niet tegelijk van de context links en rechts afhangen vanwege optimale substructuur (zie Appendix B).

In de andere tokenisers met viterbisegmentatie (Morfessor en ULM) gold de unigramaanname, m.a.w. dat de kans op elk token niet geconditioneerd was op de rest van het woord (Morfessor) of de zin (ULM). Let goed op: dat wil niet zeggen dat die rest geen rol speelt in de volledige segmentatie, want het hele punt van viterbi is dat het een globaal optimum vindt i.f.v. alle mogelijke tokens. Met andere woorden: in zowel ULM als DPE hangt de segmentatie van elk woord af van de omringende woorden, maar in ULM hangen de kansen gebruikt in die segmentatie níét van de omringende woorden af.

BPE behandelt elk woord in vacuüm. Vandaar ook dat Sennrich e.a. (2016) het trainingsproces van BPE laten voorafgaan door een compressie van het hele trainingcorpus tot (woord,aantal)-paren. Zinscontext gaat compleet verloren. Dat is onacceptabel in talen zoals het Hebreeuws, zoals reeds vermeld in §4.4, waarin zowel woordinterpretatie als woordsegmentatie sterk afhangen van de context. Klein en Tsarfaty (2020) geven als voorbeeld het woord **סלצב** dat afhankelijk van de context "in hun schaduw", "hun ajuin" of "in de fotograaf" kan betekenen.

Kumar en Thawani (2022) laten BPE overheen spaties mergen om na te gaan of betekenisvolle MWE's (bv. stadsnamen met meer dan twee woorden) in het vocabularium terechtkomen. Het antwoord is neen: door BPE spaties te laten negeren, kan het in zekere zin de context zien, maar dat "zien" is altijd beperkt tot aangrenzende tokenparen. Vergelijk dat met ULM of DPE, waarin woorden aan weerszijden van een zin beide bijdragen tot het globale optimum.

4.7 Infereren zoals leren (He e.a., 2020)

Samengevat – BPE neemt aan dat zijn gebruiksfase zich moet spiegelen aan de leerfase en dier historiek. Die aanname is beperkend en ongemotiveerd.

BPE speelt tijdens de gebruiksfase de merges opnieuw af in historische volgorde, naar analogie met Gage (1994). Vanuit het perspectief van compressie is het uiteindelijke doel om de trainingdata te coderen, dus spreekt het voor zich om de geleerde volgorde ook toe te passen. Wat nu met bijkomende data? Het algoritme van Gage was bedoeld voor de compressie van individuele bestanden, en helemaal niet voor extrapolatie naar data daarbuiten. Door dat wel te doen, veinzen we dat die nieuwe data *wel* bij de oude data hoorde en er samen mee gecomprimeerd is (op lijn 8), maar tegelijk telt de nieuwe data *niet* mee voor het bepalen van de volgende merge (op lijn 7). Door die asymmetrie is het niet vanzelfsprekend dat we merges in historische volgorde zouden afspelen.

De hele filosofie van compressie staat overigens haaks op morfologie: om een nieuwe string te segmenteren tracht BPE om het aantal tokens te minimaliseren – het stopt pas met mergen wanneer er geen enkele merge meer herkend wordt. Het doel van subwoordsegmentatie is juist om tokens níét te groot te maken en zo te kunnen veralgemenen overheen woorden. BPE-dropout kan vroeger stoppen met mergen, maar er is geen garantie dat de *alternatieve* merges uiteindelijk ook voor *minder* merges zorgen. Morfessor

Cat-MAP heeft wel de juiste filosofie: merge juist genoeg om betekenisvolle tokens te krijgen, en niet meer (cfr. §4.10).

Voor zover ik weet tonen He e.a. (2020) als eerste dat vocabularisatie en inferisatie in BPE niet per se moeten samenvallen. Dergelijke modulariteit is duidelijk voordelig: op die manier kunnen we zelf het beste vocabularisatiealgoritme kiezen en het combineren met het beste inferisatie- en segmentatiealgoritme. Jammer is wel dat DPE alleen werkt in de decoder van een NMT-systeem, en niet in een MLM (BERT) noch CLM (GPT): voor MLM werkt het niet omdat de bidirectionaliteit de optimale substructuur van het viterbialgoritme schendt, en voor CLM – een unidirectionele decoder zonder extra informatie van een bidirectionele encoder – tonen He e.a. dat DPE even slecht presteert als BPE. Het probleem is kennelijk dat DPE wel degelijk nood heeft aan bidirectionele informatie, maar tijdens het segmenteren slechts unidirectioneel kan redeneren.

Desalniettemin kunnen we voor taalmodellen andere segmentatiemethodes bedenken gegeven een bestaand vocabularium V (bv. gegenereerd met BPE), zonder de BPE-mergehistoriek te gebruiken of hiërarchisch te zijn (i.e. dat er tijdens het segmenteren tokens ontstaan die ook terug verdwijnen als onderdeel van grotere tokens). §5.2 verkent enkele greedy alternatieven.

4.8 Dataonbalansfragiliteit

Samengevat – BPE focust zich op het talrijkste gedeelte van zijn trainingcorpus, en krijgt zo een vervormd beeld van de corpustaal/-talen.

4.8.1 Meertalige corpora

Systemen die met meer dan één taal omgaan – alle TM's, en meertalige LM's zoals mBERT (Devlin e.a., 2019) en XLM (Lample en Conneau, 2019) – hebben meertalige trainingcorpora. Om die te segmenteren kan men ervoor kiezen om taalspecifieke tokenisers te trainen, of één grote tokeniser overheen alle talen samen, hetgeen Sennrich e.a. (2016) *joint BPE* noemen voor BPE.

Verschillende bronnen geven echter aan dat BPE ongeschikt is in "joint"-modus, omdat het niet kan balanceren tussen de verschillende talen. Huck e.a. (2017) mijden joint BPE opdat de tokeniser niet zou beginnen te hallucineren over Duitse morfemen in het Engels. Ook schrijven ze dat er geen controle is over de onbalans in het vocabularium:

> [...] in joint-BPE, vocabulary sizes cannot be controlled independently on each side. Joint-BPE with 59 500 operations [...] yields 46K German types in the data, but an English corpus containing only 26K types.

Ács e.a. (2021) gaan akkoord:

> [...] the number of subword tokens is highly dependent on the language. English words are only split 16.9% of the time [...], while in many other languages more than half of the words are tokenized into two or more subword units. We hypothesize that this is due to the combination of the characteristics of the English language and its overrepresentation in the training data and the subword vocabulary.

Chung e.a. (2020) voegen daaraan toe dat het populairste alfabet in een corpus met meer dan twee talen sterk bevoordeeld is:

> [...] subwords in common scripts like Latin and Cyrillic have a higher chance of selection since their counts are combined across a large number of languages. [...] the joint procedure

> over-emphasizes cross-lingual subword sharing – even across languages with little lexical overlap [...]. It also under-emphasizes [...] low-resource languages or those written in scripts used by few languages [...] to contribute subwords that are most effective [...]

Wat in dat opzicht zeker niet helpt is dat Sennrich e.a. (2016) voorstellen om een taal als Russisch te translitereren (bv. президент → *prezident*) vooraleer er joint BPE op toe te passen. Lample en Conneau (2019) promoten joint BPE om verwante talen elkaar te doen aansterken:

> For low-resource languages, it is often beneficial to leverage data in similar but higher-resource languages, especially when they share a significant fraction of their vocabularies. [...] Nepali [...] and [...] Hindi [...] have more than 80% of their [Wikipedia] tokens in common in a shared BPE vocabulary of 100k subword units.

In hun voorbeeld is er zesmaal meer data voor Hindi, dus hun suggestie komt neer op het gebruiken van een tokeniser voor Hindi in het Nepalees. Om joint BPE gebalanceerder te maken, stellen Chung e.a. (2020) voor om talen te clusteren en joint BPE op elke cluster apart toe te passen. Daartoe stellen ze elke taal met een vector voor: eerst leren ze een BPE-vocabularium over elke taal apart, dan creëren ze een one-hot-codering (cfr. §2.3) voor de unie van alle subwoordvocabularia, en tot slot nemen ze voor elke taal de som van de one-hot-vectoren van haar subwoorden.

4.8.2 Corpora met te specifieke domeinen

Het doel van een NLP-model is om "de taal" te verstaan zoals mensen. Mensen die dezelfde taal spreken, komen echter niet per se met dezelfde woordenschat in contact: inderdaad, studenten ingenieurswetenschappen en geneeskunde, twee *domeinen*, gebruiken veel woorden die in slechts een van beide disciplines bekend zijn. Als we zonder verdere specificatie "de taal" willen leren, dan gaat dat veeleer over de intersectie van alle domeinen dan hun unie. Meer over domeinspecificiteit in §4.9.

LM's voor algemeen gebruik (bv. BERT en GPT) hebben dus best een vocabularium dat niet te sterk gefocust is op domeinspecifieke taal. Subwoordtokenisers hebben net als woordtokenisers een beperkte capaciteit $|V|$, en die is noodgedwongen kleiner dan het lexicon van de taal. Om een indruk van schaal te geven: BPE_{nl} heeft 40 000 subwoordtypes, terwijl e-Lex meer dan 200 000 lemmata bevat (cfr. §C.2). Uit de crowdsourcingexperimenten van Keuleers e.a. (2015) en Brysbaert e.a. (2016) volgt dat Nederlandstaligen en Engelstaligen tussen de 30 000 en 40 000 resp. 50 000 lemmata beheersen. Elk subwoordtype is dus kostbaar.

BPE_{nl} is getraind op het Nederlandse deel van OSCAR, afkomstig van terabytes tekst vanop het web (cfr. §C.1). Het web staat gekend voor de alomtegenwoordigheid van pornografische websites, dus is het mogelijk dat tokenisers die hun vocabularium op internetdata leren een overrepresentatie aan vulgariteiten kennen. Om die hypothese te bevestigen, zoek ik doorheen een zelf ingezamelde lijst van vulgaire woorden (zie §C.4) naar de subwoordtypes van BPE_{nl}. Voor types met SOW is de zoektocht beperkt tot het begin van elk woord in de lijst. Als het subwoord exact in de lijst staat, telt het sowieso mee; als het woord in de woordenlijst strikt groter is, vereis ik dat het type minstens 5 karakters heeft om geen te algemene subwoordtypes terug te krijgen.
Dat proces resulteert in 450 matches. Voor elk gematcht subwoordtype zoek ik het meest frequente woord in OSCAR op dat dat subwoordtype bevat, en check ik manueel of het inderdaad pornografisch is of niet. Sommige types zijn dat verrassend genoeg niet: zo is het frequentste woord in OSCAR dat het subwoord *agina* bevat uiteraard gewoon *pagina*, en het subwoord *ootje* is te vinden in het onschuldige woord *pootjes*. Een raadsel voor de lezer: waar komen de BPE_{nl}-subwoorden *ampie*, *jacul* en *hemale* vandaan?

Na manuele filtering blijven er 180 types over, die ik voor de onverschrokken lezer in Tabel D.2 heb afgedrukt. Ze zijn geordend a.d.h.v. hoeveel keer ze als substring in e-Lex-lemmata voorkomen, waardoor de subwoordtypes "minder Nederlands" worden naarmate de tabel vordert.

Noteer dat het bovenstaande experiment zeer conservatief is: er zijn waarschijnlijk nog veel meer subwoorden van lengte ≤ 4 afkomstig van hetzelfde deel van OSCAR – immers, die van lengte 5 moeten op een of andere manier gevormd worden, zie §4.10. Bovendien merkte ik bij nadere inspectie dat het meest frequente OSCAR-woord vaak onschuldig was, maar het tweede of derde pornografisch. *ootje* is dan toch niet zo onschuldig.

4.9 Domeinmaladaptatie (El Boukkouri e.a., 2020)

> **Samengevat** – BPE overfit op zijn trainingcorpus. Toegepast op een ander corpus (i.h.b. over een ander onderwerp) is de segmentatie rommeliger. Bovendien is er geen manier om een BPE-tokeniser te fine-tunen.

In de vorige sectie bleek een BPE-tokeniser tijdens het leren nogal snel afgeleid door te dominante domeinspecifieke tekst, wat een probleem vormt voor algemenetaalmodellen. Wat als we echter een domeinspecifiek model willen? Beltagy e.a. (2019) vervangen het trainingcorpus van BERT (Devlin e.a., 2019), dat bestond uit Engelse literatuur en Wikipedia, door ongeveer evenveel tekst uit Engelse wetenschappelijke papers. Ze gebruiken dat corpus *zowel* voor het trainen van het model *als* het leren van BERT's WPM-tokeniser. Het resulterende model, SciBERT, deelt minder dan de helft (42%) van zijn subwoordtypes met BERT, en moet dus pre-trainen vanaf nul gezien er voor de meeste subwoordtypes geen bestaande embedding in BERT aanwezig is en ze bovendien met andere merges samenhangen.

Die aanpak is echter bijzonder duur, daar ze ingaat tegen de filosofie van transfer learning (het hergebruiken van reeds uitgevoerde training); Engelse papers volgen namelijk nog steeds de grammatica van de Engelse taal, waarvan de kennis reeds in BERT zit. Lee e.a. (2020) pre-trainen een BERT-model, BioBERT, over een corpus van biomedische papers, maar hergebruiken de tokeniser van BERT waardoor ze hun pre-training kunnen initialiseren met de eindtoestand van BERT.

Dan is de kennis van BERT wel hergebruikt, maar El Boukkouri e.a. (2020) tonen a.d.h.v. een medisch corpus dat domeinspecifieke woorden significant meer tokens produceren bij gebruik van zo'n domeinagnostische tokeniser (zie hun figuur 1) en dat de splitsingen weinig betekenisvol zijn (zie hun tabel 1), hetgeen in §4.13 terugkomt. Ook Amrhein en Sennrich (2021) merken dat op:

> […] even slight deviations from the text seen when learning a segmentation model can result in entirely different segmentations and often aggressively over-segmented text.

Idealiter kunnen we én de gewichten van een model na pre-training gebruiken, én een tokeniser gebruiken die aan het domein is aangepast; m.a.w. wensen we een soort fine-tuning van de tokeniser op het domeinspecifieke corpus. De vrijheidsgraden in een BPE-tokeniser zijn de types V en de merges M. Dat is een probleem, want fine-tuning slaat normaliter op het bijstellen van parameters met continu domein, met name tegen de gradiënt van een verliesfunctie in; het verwijderen of toevoegen van types is daarentegen een discrete operatie zonder verliesfunctie.

Daarbij is het niet evident om een type te verwijderen uit de mergeboom, gezien zijn kinderen dan een ouder missen. Een type toevoegen is makkelijker, maar hoe beslissen we welk type, welke merge het tot stand brengt, en welke prioriteit die heeft in M?

Immers, elke merge in de BPE-graaf is geconditioneerd op alle voorgaande merges, zoals SciBERT toonde. Vandaar is het ook moeilijk om twee BPE-mergegrafen, bv. de ene voor het oorspronkelijke corpus en de andere voor het domeinspecifieke corpus (of de unie van de twee), samen te voegen.

Ter illustratie: Delobelle e.a. (2022) trainen een tweede BPE-tokeniser voor RobBERT op een uitgebreide versie van OSCAR, en proberen die dan te combineren met de bestaande BPE-tokeniser. Ze selecteren alle types die nog niet in V zaten, en voegen die toe aan V, waarbij ze de respectieve merges in volgorde en *helemaal achteraan* M toevoegen – d.w.z. met laagste prioriteit.[54] Vandaar begint de nieuwe segmentatieboom van *coronamaatregelen* (Figuur D.9) met alle oude merges (Figuur D.8).
Omdat de nieuwe merges echter niet geleerd zijn in hetzelfde iteratieve proces als de rest van de mergegraaf, kan het zijn dat ze nooit aangesproken worden. Een abstract voorbeeld: stel dat de oude en nieuwe BPE-tokeniser beide leren om de subwoordtypes A, B en C te vormen. De oude leert bovendien de merge tot AB, terwijl de nieuwe de merge BC leert, maar niet (of pas later) AB. Wat is nu de segmentatie van een woord ABC? De oude tokeniser zegt $AB|C$, de nieuwe tokeniser zegt $A|BC$, en door de nieuwe merges de laagste prioriteit te geven zegt hun combinatie nog steeds $AB|C$. Sterker nog: als de nieuwe tokeniser voortbouwt op dat type BC en bijvoorbeeld ABC aanleert, *dan nog* wordt de string ABC als $AB|C$ gesplitst! De reden is enerzijds omdat de merge $A + BC$ geconditioneerd is op de merge $B + C$, en anderzijds dat alle merges in een BPE-tokeniser een uniek product hebben (zie §4.17), waardoor de nieuwe dus niet óók $AB + C$ kan leren na $A + BC$.

Om die abstracte hypothese te testen, volstaat het dus om de nieuw toegevoegde subwoordtypes tussen RobBERT-2020 en RobBERT-2022 om te zetten tot strings (waarbij de SOW vervangen wordt door een spatie), en dan te laten segmenteren door RobBERT-2022. Als de hypothese klopt, dan zouden die strings soms moeten resulteren in meer dan één token, zelfs al zijn ze als volledig type aanwezig in het vocabularium. Figuur D.11 bevestigt dat dat het geval is.

Andere segmentatiemethodes maken het gemakkelijker. Tai e.a. (2020) stellen exBERT voor met een WPM-model dat segmenteert door van links naar rechts het grootste gekende token af te splitsen. Eerst leren ze een tweede WPM-tokeniser overheen eender welk domeinspecifiek corpus, en voegen net als Delobelle e.a. (2022) de missende types toe aan V, alleen moet hun segmentatiemethode geen merges onthouden. Om de bijkomende embeddings alsook de mogelijke verandering in zinsstructuur te leren, voegen ze aan elk van de N_e transformermodules (cfr. Figuur 2.1) een kleine extra module toe die de zin in parallel verwerkt. De resulterende vector van elk token is een gewogen som van de oude en nieuwe module, met het gewicht bepaald door een dense neurale laag.

Natuurlijk is het aangenamer om architecturen te hanteren waarin het fine-tunen van de segmenteerder triviaal is. Zo zagen we in §2.4.1 hoe de statistische samenstellingssplitser van Macken en Tezcan (2018) adaptieve frequenties had, en in § 2.5.4 hoe de tokeniser in het model verwerkt zat in karaktergebaseerde end-to-endmodellen.

[54] Dat laatste heb ik niet uit de paper maar uit persoonlijke correspondentie met P. Delobelle.

4.10 Afval (Bostrom en Durrett, 2020)

> **Samengevat** – BPE werkt bottom-up: vooraleer toegang te hebben tot langere, betekenisvolle subwoorden, moet het V opvullen met kleine betekenisloze subwoorden.

Om toegang te krijgen tot een subwoord van lengte n, zijn er in het beste geval – wanneer de mergeboom een perfect gebalanceerde binaire boom is – $\lceil \log_2 n \rceil$ types nodig als tussenstap naar dat subwoord, en in het slechtste geval – één karakter per iteratie – de volle n tussenstappen. Dat laatste is vergelijkbaar met het welbekende *kettingeffect* dat agglomeratieve clustering teistert.

Of het nu $\lceil \log_2 n \rceil$ of n is, het is hoe dan ook verspilling om V met dergelijke tussenstappen te vullen. Het gevolg is in feite dat de middelgrote subwoordtypes, nl. die tussen karakters en morfemen en tussen morfemen en woorden, geen betekenis hebben. Ze nemen kostbare capaciteit in die voor nuttigere subwoorden had kunnen dienen, met name meer unieke woordstammen. Bostrom en Durrett (2020)[55] tonen (in hun figuur 2) dat het vocabularium van BPE t.o.v. ULM veel meer types bevat die quasi nooit als token voorkomen in het corpus dat met de finale tokeniser gesegmenteerd is. Tijdens vocabularisatie zijn ze uiteraard wel aanwezig – anders waren ze nooit gekozen door lijn 7 – maar ze worden slechts als tussenstappen gebruikt en volledig geconsumeerd door grotere merges. ULM heeft dat probleem niet aangezien het top-down werkt (cfr. Algoritme 2.3): alle mogelijke subwoorden zijn in de eerste iteratie reeds beschikbaar, en subwoorden verwijderen gebeurt ongestructureerd.

Morfessor Cat-ML en Cat-MAP (Creutz en Lagus, 2005a) lossen een verwant afvalprobleem op: ze proberen subwoorden te classificeren in de drie categorieën prefix, woordstam en suffix (cfr. §2.5.2.4), maar dat gaat niet voor subwoorden kleiner dan morfemen aangezien die voorkomen als deel van elke categorie. Vandaar hebben ze een "afvalcategorie" voor die ambigue subwoorden, waarmee ze zichzelf bekendmaken. Om dergelijke subwoorden te mijden tijdens segmentatie, en aangezien al die kleine subwoorden onderaan in de boom zitten (cfr. Figuur 2.3), verbieden ze hun recursieve tokeniser om een subwoord nog verder te segmenteren als dat voor minstens één afvalsubwoord zou zorgen. Er is dus een neerwaartse druk die zoveel mogelijk wil splitsen, en een opwaartse druk die de subwoorden betekenisvol houdt.

Salesky e.a. (2018) merken afvaltypes anders dan Bostrom en Durrett (2020), nl. in de uitvoer van hun NMT-model i.p.v. in de invoer. Telkens wanneer iBPE meer subwoordtypes vrijgeeft om te produceren, geraken er een heleboel bestaande types in onbruik:

> After moving to the 20k vocabulary, 26% of [subword types in the tuning set] are no longer generated. Similarly, when incrementing from 20k to 30k, 26% of the previous embeddings are no longer generated. In addition, we see that ∼50% of the "obsolete" subwords are also not present in the equivalent references for the next increment. […] these numbers suggest some are no longer needed and could reasonably be dropped.

Hun voorgestelde oplossing houdt evengoed steek voor BPE als voor iBPE: zoals Figuur 1.1 toonde, is een model zich niet bewust van $|V|$ aangezien het domein van de invoer continu is, en dus kan het moeilijk herkennen wanneer een subwoord voorgoed verdwijnt in de invoer (en bv. niet gewoon een lichtjes geperturbeerde embedding heeft). De enige componenten waarin $|V|$ ingebakken zit, zijn de embedding-LUT vóór en de MLM- of CLM-softmaxlaag bovenop het model, die men meestal gelijkstelt om parameters te besparen (Vaswani e.a., 2017). Daar ze gewoon matrices van $|V|$ gescheiden embeddings zijn, kan men subwoorden afzonderlijk laten vallen. §5.3 verkent dat meer.

[55] *"Byte Pair Encoding is Suboptimal for Language Model Pretraining"*

4.11 Datainversie (Provilkov e.a., 2020)

Samengevat – Kleinere subwoorden zijn belangrijker dan grotere, en terwijl ze als string meer in het corpus voorkomen, komen ze als token minder voor in de trainingdata en worden zo minder goed geleerd.

Van welke subwoorden leert een taalmodel de rol in de taal het best? Intuïtief, en ook volgens de DH, zijn het die die het meest voorkomen in de teksten van het trainingcorpus. Dat zijn niet alleen de meest gebruikte woorden in de taal (lidwoorden, voornaamwoorden, alledaagse woordenschat ...), maar ook de morfemen die gedeeld worden door alle woorden om verbuigingen te ondergaan. Dat was immers een van de argumenten voor subwoordtokenisatie in §1.4: zelfs ongeziene woorden zijn alsnog te interpreteren op basis van hun gedeelde morfemen, en dat zou juist heel goed moeten gaan omdat dat de subwoorden zijn die het taalmodel meer dan alle andere subwoorden heeft waargenomen.

BPE gaat tegen die intuïtie in. De frequentste subwoorden worden door BPE het snelst ontdekt en gemerged, maar dat betekent ook dat ze vervolgens systematisch deel uitmaken van nog grotere merges. Het resultaat is dat de belangrijkste types in het vocabularium (morfemen), die als string zeer frequent in het corpus voorkomen, juist zeer weinig als token voorkomen, gezien ze worden opgeslokt door grotere tokens. Volgens Provilkov e.a. (2020):

> [...] one of the drawbacks of BPE [is that] only rare words are split into subwords. This forces frequent sequences of characters to mostly appear in a segmented text as part of bigger tokens, and not as individual tokens.

Ook Bostrom en Durrett (2020) vermelden het in een voetnoot:

> [...] the BPE vocabulary still includes [...] affixes, but when they are encountered during tokenization, they are almost always merged into larger units [...]

Er is geen expliciete communicatie tussen de kennis die een taalmodel opdoet over een subwoordtype en de kennis over zijn ouders – de twee types waaruit het bestaat – aangezien een taalmodel niet weet waar subwoordtypes vandaan komen of wat hun onderliggende tekst is (cfr. Figuur 1.1), i.t.t een menselijke lezer. Er is dus ronduit dataverlies: de informatie die in grotere types vervat zit over kleinere types, geraakt niet tot bij de kleinere types. Dat is duidelijk een groot probleem, want het zijn juist die kleine types die we zo goed mogelijk willen beheersen om ⟨UNK⟩'s op te vangen!

Om de bovenstaande hypothese te staven, meet ik voor elk type in het vocabularium van BPE_{nl} het aantal keer dat het in OSCAR als (sub)string in de tekst voorkomt f_s, en het aantal keer dat het als token voorkomt na segmentatie, f_t. §D.6 toont enkele relevante figuren.

Ten eerste geeft Figuur D.12 een scatterplot van elk (f_s, f_t)-paar: noteer dat een log-logschaal gebruikt wordt aangezien taal een zipfiaanse verdeling volgt, en de absolute frequenties dus zeer uiteenliggende ordegroottes hebben. De 45°-lijn getuigt van een correcte telling, daar de aanwezigheid van een token altijd die van de onderliggende substring impliceert, maar omgekeerd niet altijd, i.e. $f_s \geq f_t$. Er is datainversie tussen twee subwoordtypes wanneer de ene meer in de tekst voorkomt maar de andere meer als token, m.a.w. $f_{s,1} > f_{s,2}$ maar $f_{t,1} < f_{t,2}$. Voor een gegeven type (f_s, f_t) vinden we de verzameling van daartegenover geïnverteerde types $\{(f'_s, f'_t) \mid f'_s < f_s \wedge f'_t > f_t\}$ in de rechthoekige driehoek met de 45°-lijn als hypotenusa en (f_s, f_t) als rechtehoekpunt. Datainversie is duidelijk talrijk aanwezig: van de $\binom{40\,000}{2} = 799\,980\,000$ typeparen waarvoor $f_{s,1} > f_{s,2}$ zijn er 253 538 996 ($\sim 32\%$) waarvoor toch $f_{t,1} < f_{t,2}$.

Er zit redundantie in het scatterplot aangezien de horizontale en verticale afstand tot de 45°-lijn gelijk is. Die afstand is $\log_{10}(f_s) - \log_{10}(f_t) = \log_{10}(f_s/f_t)$, is evenredig met de rechte afstand van een type tot de 45°-lijn, en lijkt een goede maatstaf te zijn voor de mate van dataverlies van een type. Figuur D.13 toont de verdeling van die verhouding. Op het scatterplot van $(f_s, f_s/f_t)$ verschijnt voor $f_s > 10^4$ een andere schuine lijn met helling 1: dat is een hele groep tokens waarvoor de wet $\log_{10}(f_s/f_t) = 1\log_{10}(f_s) + c$ geldt, m.a.w. vertonen al die types hetzelfde aantal tokens terwijl de vereiste hoeveelheid tekst om dat aantal te bereiken exponentieel toeneemt. Op deze grafiek is er trouwens opnieuw een bovengrens (die waar $f_s/f_t = f_s$, evenwijdig met de schuine lijn in de data) die de punten opvallend genoeg pas beginnen te benaderen voor hoge f_s.

We weten nu dat BPE voor datainversie en dataverlies zorgt, en dat het erger wordt voor veelvoorkomende substrings. Wat met morfemen, waar we zeker geen dataverlies voor willen? We weten dat morfemen kleine strings groter dan letters zijn (zie Figuur D.1): het is dus interessant om de verdeling van de f_s/f_t-verhouding te groeperen per typelengte. Figuur D.14 toont ze in de vorm van boxplots. Helaas: het zijn juist de kleine types groter dan letters die het meeste verlies lijden.

De ideeën van iBPE zijn een beginnende oplossing. iBPE toont dat kennis over subwoorden niet alleen kan komen van training over het corpus, maar ook van reeds aanwezig kennis over andere subwoorden in het model. Salesky e.a. (2018) motiveren iBPE als een manier om informatie door te geven van ouders (kleine subwoorden) naar kinderen (grote subwoorden) om zo te compenseren voor het feit dat strings minder dan hun substrings voorkomen, maar mijns inziens is het door datainversie juist *omgekeerd*: iBPE houdt grotere merges aanvankelijk tegen, dus lijden de kleine subwoorden tijdelijk niet onder datainversie waardoor ze deftig uit het corpus aangeleerd kunnen worden. Salesky e.a. redeneren vanuit strings, terwijl de trainingdata uit tokens bestaat.

Het delen van informatie in iBPE is echter nog te beperkt: immers, de ouders van een subwoordtype zijn niet de enige informatieve types in het vocabularium. Neem nu de mergegraaf van Figuur 1.2: *coronamaatregelen* (bovenaan midden) zou er duidelijk baat bij hebben om bv. gebruik te maken van de embedding van *corona* (bovenaan rechts), maar dat is een kozijn, geen ouder. In dit voorbeeld is het nog oplosbaar door betere ouders te selecteren (*corona* i.p.v. *coron*), maar algemeen kan het nuttig zijn om over langere paden doorheen de BPE-mergegraaf te communiceren. Een andere beperking van iBPE is dat het delen in de foute richting gebeurt: inversie biedt de gróte subwoorden juist meer trainingdata, dus zouden ouders (kleine subwoorden) juist van hun kinderen moeten profiteren.

4.12 Vocabulariumgrootteresearchbias en -paradox

> **Samengevat** – BPE is een iteratief algoritme zonder dynamische stopconditie. Niemand weet wat de beste BPE-vocabulariumgrootte is. Researchers nemen elkaars hyperparameters over, zelfs bij de overstap van LSTM's naar transformers. Bovendien werkt BPE bottom-up, waardoor het zich niet tegelijk kan beperken tot kleine morfemen maar grote lexicaliseringen.

4.12.1 Optimale vocabulariumgrootte

De grootte van het vocabularium $|V|$ is een vrij te kiezen hyperparameter τ bij het leren van de BPE-tokeniser (zie Algoritme 2.1). Aangezien τ voor volledig andere tokenisaties zorgt aan de invoer van het neurale model achter de tokeniser, vereist elke keuze van τ een hertraining van het model vanaf nul, waardoor τ gezien kan worden als een

hyperparameter van het model zelf.

I.t.t. andere machinelearningarchitecturen is het voor grote neurale modellen zeer duur om hyperparameters te tunen; niet alleen omdat dergelijke modellen inherent vele iteraties met vele berekeningen vergen, maar ook omdat ze veel verschillende vrijheidsgraden hebben (aantal neurale lagen, aantal neuronen per laag ...) met veel mogelijke waarden. Tuning is nodig om zeker te zijn dat de gekozen hyperparameterwaarden niet tegenwerken. Een extra hyperparameter is dus een zwaar nadeel.

Veel iteratieve algoritmes blijven itereren tot een zekere dynamische, ongeparametriseerde convergentie bereikt wordt: k-means eindigt wanneer de clusters niet meer veranderen van de ene iteratie op de andere, de newton-raphsonmethode eindigt wanneer de relatieve afwijking van het resultaat van de voorbije twee iteraties kleiner is dan machineprecisie, enzovoort. Het is juist eerder ongewoon om zoals BPE het aantal iteraties manueel vast te leggen.

4.12.1.1 Heuristieken

Verschillende papers zoeken naar heuristieken om τ te bepalen zonder te moeten tunen. Ding e.a. (2019)[56] doen een groot empirisch onderzoek naar hoe BLEU-scores variëren in functie van τ, en dat telkens geparametriseerd door een combinatie van architectuur (LSTM of transformer, ondiep of diep) en taalpaar (Arabisch, Tsjechisch, Duits, Frans, van of naar Engels). Hun conclusie is dat transformers en diepe architecturen het altijd beter doen met kleinere vocabularia ($\tau \in [500, 1000]$), en dat het verschil groter wordt naarmate het corpus kleiner is – logisch, want dan krijgen de modellen niet de kans om zeldzame woorden te memoriseren.

Gowda en May (2020) doen iets gelijkaardigs, alleen hebben zij een vaste transformerarchitectuur: ze ontdekken de heuristiek dat de optimale τ vaak ligt waar de 95-percentiel van tokenfrequenties (i.e. de frequentie van het type dat minder voorkomt dan 95% van alle types) ongeveer 100 is. De reden om dat getal niet gewoon zo hoog mogelijk te maken (en dus meer trainingdata te hebben zelfs voor de zeldzame subwoordtypes) is omdat het vocabularium dan, zoals ook Ding e.a. (2019) en Amrhein en Sennrich (2021) vaststellen, wel degelijk te kléín wordt.

Tot slot tonen Xu e.a. (2021) dat de negatieve afgeleide van tokenentropie $\mathcal{H}$ (zie Vgl. 2.10) naar τ, d.w.z. de afname in entropie door toename in vocabulariumgrootte $-\frac{\partial \mathcal{H}}{\partial \tau}$, maximaal is wanneer BLEU ook maximaal is i.f.v. τ. Ze discretiseren de afgeleide tot een differentiequotiënt, de *marginal utility of vocabularisation (MUV)*:

$$\text{MUV}(\tau) = -\frac{(\mathcal{H}(\tau + \Delta\tau) - \mathcal{H}(\tau))}{\Delta\tau} \qquad \text{met } \tau \in \mathbb{N} \text{ en } \Delta\tau \in \mathbb{N}. \tag{4.1}$$

Het maximaliseren van de MUV is dus ook een heuristiek. In plaats van een hele reeks NMT-modellen te trainen, volstaat het om een sequentie van τ-waarden op te stellen op afstand $\Delta\tau$ van elkaar (bv. 1k, 2k, 3k ...), en om vervolgens voor elke waarde een BPE-tokeniser te leren, die het corpus te laten segmenteren, daarvan de tokenfrequenties te tellen, daarmee de entropie te berekenen, die van elkaar aftrekken om de MUV te krijgen, en daar dan het maximum van te vinden. De heuristiek van Gowda en May (2020) kost nagenoeg evenveel.

[56] *"A Call for Prudent Choice of Subword Merge Operations in Neural Machine Translation"*

4.12.1.2 Automatische methodes

Xu e.a. (2021) geven een tweede manier om te vocabulariseren, dit keer zonder BPE. Enerzijds staan ze toe om MUV te berekenen tussen arbitraire vocabularia van grootte τ en $\tau + \Delta\tau$ i.p.v. alleen die gegenereerd door BPE, en anderzijds vervangen ze het zoeken naar maximale MUV door het zoeken naar maximale entropie.[57] Die probleembeschrijving zetten ze om naar de algemene vorm van een probleem in *optimaal transport* (verwant aan LP-problemen) waarvoor er bestaande oplossers zijn. De volledige methode heet *vocabulary learning via optimal transport (VOLT)* en verloopt dan als volgt: eerst genereren ze een grote verzameling strings (gelijkaardig aan ULM) en wijzen ze aan elk karakter van het alfabet een budget toe o.b.v. hun voorkomen in het corpus. Vervolgens laten ze elk budget door de oplosser uitdelen aan de strings die dat karakter bevatten. Het vocabularium bestaat uiteindelijk uit de τ strings met het grootste toegewezen budget. Aangezien VOLT geen merges teruggeeft, vereist het een aparte inferentiemodule (bv. DPE) zoals besproken in §4.7.

MUV en VOLT berekent men offline, d.w.z. voorafgaand aan modeltraining. iBPE (§2.5.2.2) laat zich in plaats daarvan leiden door de online-veranderingen in modelverlies over een tuningcorpus van de ene epoch op de andere. In zekere zin bepaalt het τ automatisch, maar τ fungeert hier eerder als een iteratienummer dan een iteratielimiet. τ stopt met stijgen wanneer het model stopt met trainen, en dat gebeurt wanneer het modelverlies 10 epochs op rij stijgt.

Een van de redenen waarom BPE geen convergentiecriterium heeft, is dat de BPE-metriek (frequentie) niet genoeg informatie geeft over de meerwaarde van nog meer types toe te voegen, o.a. omdat frequentie discreet is en de wet van Zipf volgt. Dat is niet zo voor de metriek van S-BPE (§2.5.2.1), waardoor het wel een voorspelbaar convergentiecriterium heeft: noem de waarde van Vgl. 2.15 δ_i op iteratie i, dan is de rij $\delta_1, \delta_2, \ldots, \delta_\tau$ monotoon dalend net als de frequentie in BPE. Vilar en Federico (2021) stellen voor dat τ die iteratie is waarvoor voor het eerst $\delta_i/\delta_1 \leq \alpha$ met $\alpha < 1$. Bij tuning blijkt die α keer op keer dezelfde optimale waarde te hebben ($\alpha = 0.002$), dus kan men die waarde gewoon overnemen. Zelfs met die vaste waarde heeft S-BPE significant betere BLEU dan 4k-, 8k-, 16k-, 32k-, 48k-, 64k-, en 96k-BPE, zowel door de betere metriek (en dus betere subwoorden) als de exacte τ.

4.12.2 Researchbias (Ding e.a., 2019)

Nu wordt de historische context van §2.1 van groot belang. Herinner dat BPE door Sennrich e.a. (2016) ontwikkeld werd in de architectuur van Bahdanau e.a. (2015), bestaande uit een encoder-LSTM en een decoder-LSTM. Herinner ook dat elke LSTM een interne informatieflessenhals heeft waarbij verafgelegen woorden elkaar slechts zeer troebel zien, wat pas door Vaswani e.a. (2017) opgelost werd met transformers. Toen de BPE-paper geschreven werd, werkten onderzoekers in het kader van LSTM's, en bij die tijdsgeest hoorde o.a. dat ze wegschuwden van te lange sequenties. Huck e.a. (2017) filteren zelfs expliciet alle zinnen uit hun experimenten die meer dan 50 subwoordtokens lang zijn.

Het is dus niet verrassend dat Sennrich e.a. een gigantisch subwoordvocabularium promoten: hoe meer subwoordtypes er zijn, hoe minder tokens er gemiddeld nodig zijn om een zin te coderen. Ze nemen het magische getal $\tau = 30$k over van een paper die het op zijn beurt van Bahdanau e.a. heeft; in feite is dat reeds gevaarlijk, daar hun systeem 30k

[57] We willen in feite iets van de vorm $\max_{a,b}\{a-b\}$ berekenen, met kwadratische zoekruimte. Wat ze in plaats daarvan berekenen is $\max_a\{a\} - \max_b\{b\}$, tweemaal met lineaire zoekruimte, waar $\max_{a,b}\{a-b\}$ niet onder kan liggen.

woorden i.p.v. *sub*woorden bevatte. Op hun beurt baseren Wu e.a. (2016) zich voor de $\tau = 32$k van het bahdanausysteem van Google Translate op Sennrich e.a., terwijl ze niet eens BPE maar WPM gebruiken (waarvoor Schuster en Nakajima (2012) in de CJK-talen $\tau = 200$k voorstellen, waarvan 33k alfabet). Als kers op de taart nemen Vaswani e.a. (2017) die 32k over van Wu e.a., waarbij ze de tokeniser opnieuw verwisselen naar BPE.

Ding e.a. (2019) verzamelen 36 NMT-papers uit 2017 en 2018 en merken dat de helft tussen de 30k en 50k werkt.[58] Van de rest zit meer dan de helft daarboven, zonder consensus. Zoals hierboven reeds aangehaald tonen Ding e.a. vervolgens dat transformers juist veel beter presteren met τ tussen 500 en 1000, en zoals vermeld in §4.13 bevestigen Amrhein en Sennrich (2021) dat transformers alleen met $\tau = 500$ bepaalde vaardigheden hebben. Het is m.a.w. compleet ongegrond om een hyperparameter van woord-LSTM's over te nemen in subwoordtransformers.

4.12.3 Paradox (Clark e.a., 2022)

Opvallend is dat ook ULM en Morfessor een manueel in te stellen τ hebben, en dat zelfs de prior van Morfessor (Vgl. 2.23) expliciet geen voorkeur uitspreekt over welke waarde beter is daar $P(|V|)$ een constante is. Men zou dus kunnen opwerpen dat een gebrek aan dynamische keuze van de vocabulariumgrootte geen BPE-specifiek probleem is, maar een probleem met alle tokenisers. Echter, er speelt zich in BPE een paradoxaal effect af dat niet plaatsvindt in de andere.

We zagen in §4.10 dat BPE bottom-up werkt, waardoor grote subwoorden alleen gevormd kunnen worden uit reeds gevormde kleinere subwoordtypes. Figuur D.2 bevestigt daarbij dat langere subwoorden pas regelmatig gevormd worden naarmate τ stijgt, en tegen dan is er daartoe een heleboel afval geproduceerd zoals Bostrom en Durrett (2020) opmerkten. Daarnaast worden morfemen ook aan andere types geplakt, en krijgen we dataverlies zoals §4.11 uitwees. Figuur D.1 toont anderzijds dat morfemen relatief klein zijn; alles wijst m.a.w. in de richting van τ klein te houden.

Hebben we grotere subwoorden nodig? In §1.3.2 en §2.4.2 kwam het concept van lexicalisering aan bod: er is een eindige verzameling woorden in de taal waarvan de betekenis niet af te leiden is uit hun morfologie, zoals *krokodillentranen* en *boterham*. Aangezien die woorden niet compositioneel zijn, kan het geen voordeel hebben om ze in meerdere tokens op te splitsen. Clark e.a. (2022) verwoorden het a.d.h.v. "componeren" en "memoriseren":

> [...] we would like to [...] build models that learn how to compose words where appropriate, and memorize them where memorization is needed. Large frequency-derived vocabularies partially mitigate this problem by simply memorizing more, but language inherently requires aspects of both memorization and composition.

Opdat lexicaliseringen hun eigen subwoordtype zouden krijgen, moet τ zo groot zijn dat er regelmatig grote woorden gevormd worden. Welnu de paradox: we willen een BPE-vocabularium dat tegelijk kleine types behoudt en grote types bouwt. We moeten τ klein houden opdat morfemen niet worden opgeslokt door grotere types, maar we moeten τ groot maken opdat lexicaliseringen niet opgesplitst worden. BPE kan de kloof tussen heel kleine en heel grote types niet overbruggen zonder dataverlies en zonder afval; ULM en Morfessor zijn niet hiërarchisch en moeten die brug dus niet bouwen.

Voor zover ik weet, is er geen centrale lijst van Nederlandse lexicaliseringen beschikbaar om op te experimenteren. De morfologische analyse in e-Lex geeft geen indicatie van

[58] Huck e.a. (2017) schrijven zo dat ze hun keuze $|V| = 50$k niet verder tunen omdat ze "geen belangrijke inzichten verwachten voor kleinere waarden".

compositionaliteit, aangezien de betekenis van een woord zijn morfologische herkomst niet tenietdoet. Zo bevat e-Lex de analyses

muggenziften:	`((mug)[N],(e)[V|N.V],(zift)[V])[V]`	(4.2)
krokodillentranen:	`((krokodil)[N],(e)[N|N.N],(traan)[N])[N].`	(4.3)

Voor *boterham* bevat e-Lex dan weer geen analyse, maar dat is ook geen indicatie dat het een lexicalisering is; de samenstelling *muntenverzamelaar* en de afleiding *multiculturalisme* hebben bv. ook geen morfologische annotatie, terwijl ze duidelijke, compositionele morfemen hebben.

4.12.4 Nadelen van grote vocabularia (Stahlberg, 2020)

Er zijn nog andere redenen om de grote τ-waarden uit het verleden (30k, 60k, 90k, 200k ...) te mijden. Stahlberg (2020) somt de belangrijkste op, onafhankelijk van de tokeniser:

> [...] embedding matrices make up most of the model parameters [...]. Large models require a small batch size because they take more space in the (GPU) memory, but reducing the batch size often leads to noisier gradients, slower training, and eventually worse model performance [...]. Furthermore, a large softmax output layer is computationally very expensive. [...] Besides [...] 843K of the 875K distinct words (96.5%) occur less than 100 times in an English text with 140M running words [...]. It is difficult to train robust word embeddings for such rare words.

Chung e.a. (2021) staven de eerste stelling: hun frappantste resultaat is dat 71% van alle parameters in XLM-RoBERTa (een RoBERTa-model getraind over 100 talen) zich in de embedding-LUT van $\tau = 250$k subwoorden bevinden. Die (statische!) embeddings zijn vaak vectoren in $\mathbb{R}^{768}$; bij 8 bytes per element komt XLM-RoBERTa m.a.w. aan $8\,\mathrm{B} \cdot 768 \cdot 250\,000 = 1.43\,\mathrm{GiB}$ om alleen al de embeddings in te laden, zonder de rest van het model noch de data. Bij gebrek aan werkgeheugen vertraagt het trainen dus, en zelfs als het dat niet doet, kost het hoe dan ook meer tijd en meer data om meer gewichten te trainen.

Dat de softmax in een MLM-/CLM-classificeerder inderdaad $O(\tau)$ tijd en ruimte vergt, was al bekend toen Mikolov e.a. (2013b) word2vec trainden, en is bovendien de motivatie die Sennrich e.a. (2016) geven voor de magische $\tau = 30$k. De stelling over robuustheid wordt door Gowda en May (2020) gestaafd in NMT: des te ongebalanceerder de token-verdeling in de doeltaal,[59] des te lager de recall van zeldzame woorden.

Voor BPE zijn er nog bijkomende problemen met grote τ. Ten eerste heeft de leerfase tijdscomplexiteit $O(\tau^3)$ (aangezien $|V|$ elke iteratie met 1 toeneemt en lijn 7 $O(|V|^2)$ kost) bij gebrek aan benaderende acceleratiestructuren. Ten tweede is na een zekere τ de frequentiemetriek (gevisualiseerd in Figuur D.6) genoeg gedaald om geen deftige keuze meer te kunnen maken tussen alle mogelijke kandidaat-types. De onderste figuur heeft een lineaire iteratieschaal, en toont dat er steeds minder ruimte zit tussen de waarde van lijn 7 in opeenvolgende iteraties (m.a.w. de curve stagneert). Dat de frequenties van tokenparen na een tijd op elkaar beginnen te lijken, is normaal, aangezien de tokeniser na een tijd alle algemene morfemen gezien heeft en een hogere τ gewoon specifieke woorden memoriseert uit het ganse woordenboek van de taal. In feite is elke verdere uitbreiding van V een vorm van domeinadaptatie.

[59] Gemeten als absolute afwijking tot een uniforme verdeling: $\sum_{t \in V} |p_t - \frac{1}{|V|}|$.

4.13 Niet morfologisch (Vilar en Federico, 2021)

Samengevat – BPE heeft moeite met morfologie na te bootsen: de tokens hebben geen inherente betekenis, dus is het onduidelijk of ze evenveel waard zijn als hetzij een willekeurige segmentatie, hetzij een ⟨UNK⟩.

4.13.1 Probleem

Herinner de GH uit Hoofdstuk 1, die stelt dat betekenissen gelinkt kunnen worden op basis van letters alleen. Sennrich e.a. (2016) trekken de GH in feite door tot in het extreme: simpele frequentiepatronen van gepaarde karaktergroepen laten zogezegd toe om ongeziene woorden zinvoller te segmenteren dan op karakterniveau. Vilar en Federico (2021) trekken dat in twijfel:

> Intuitively, in order to generate new words, we would expect the sub-word units to have some linguistic meaning, so that a new word can be created in a compositional way. Being purely frequency-driven, BPE does not take this intuition into consideration [...]

Het is m.a.w. onduidelijk hoeveel verder we staan nu we elke invoer kunnen coderen (t.o.v. de ⟨UNK⟩'s van Bahdanau e.a. (2015)). Uiteraard moet er systematiek in de codering zitten, want een willekeurige segmentatie garandeert niet dat elk token in V zit; Chitnis en DeNero (2015) hebben zo'n systematische codering (cfr. § 2.1.2), maar hun tokens hebben meer dan één mogelijke onderliggende string i.t.t. BPE. In die zin is BPE een betere codering van de invoer, maar alle andere tokenisers in §2.5, bv. Morfessor en ULM, voldoen aan dezelfde eigenschap.

BBPE, dat bedoeld is om Unicode efficiënter te coderen dan BPE, heeft hetzelfde veralgemeningsprobleem ook op een lager niveau: stel dat de BBPE-tokeniser van een CJK-model nooit op Vietnamese UTF-8-codes getraind werd, dan kan BBPE de bytes van een Vietnamese zin wel als een rij tokens voorstellen, maar die tokens zullen geen karaktergrenzen respecteren aangezien de UTF-8-codes van het ene alfabet niet af te leiden zijn uit die van een ander alfabet.

Gebrek aan morfologie kan morfemen fragmenteren en vertroebelen. Ataman e.a. (2017) geven als voorbeeld dat hun bahdanaumodel het Turkse woord voor *verzekerden* vertaalt met *verzekeraars*. Door Turkse agglutinatie verschillen de twee van elkaar slechts door de aan- of afwezigheid van een passiverend morfeem *-il-*, en inderdaad, hun BPE-tokeniser plakt de *i* aan het subwoord links en de *l* aan het subwoord rechts.

4.13.2 Framework voor foutenanalyse

We kunnen het gebrek aan morfemen in segmentaties ook intrinsiek evalueren, a.d.h.v. de overlap van splitsingspunten tussen een morfologische referentie en de segmentatie die de tokeniser suggereert (Kurimo e.a., 2006).

Metrieken Neem als voorbeeld het woord *doctoraatsmiserie*. De morfeemsequentie is *doctor|aat|s|miserie*, terwijl BPE_{nl} het woord[60] splitst als *doctor|aat|sm|is|erie*. We zetten de letters onder elkaar, en vergelijken de tussenletterposities:

[60] Voor de volledigheid: wanneer het hier of ergens anders over de tokenisatie van een enkelvoudig woord gaat, omring ik het altijd eerst met spaties vooraleer het aan de tokeniser te geven. De `tokenizers`-implementatie van HuggingFace beschouwt de invoer anders namelijk als een substring binnenin een groter woord, en tokeniseert het zodoende foutief zonder SOW/EOW.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Referentie:	d	o	c	t	o	r |	a	a	t |	s |	m	i	s	e	r	i	e
Kandidaat:	d	o	c	t	o	r |	a	a	t |	s	m |	i	s |	e	r	i	e

De aanwezigheid van een splitsing is een binair label, dus de overeenkomst is te meten met precision, recall en F_1. Let echter op: het gaat hier níét over de precision en recall van Koehn en Knight (2003) in Vgl. 2.7, die de correctheid van splitsingen enkel op woordniveau meten (ofwel is de kandidaat volledig juist, ofwel is hij dat niet). Noem R de verzameling van de te vinden splitsingsposities in de referentie (hier $\{6, 9, 10\}$) en C die van de voorgestelde posities in de kandidaat (hier $\{6, 9, 11, 13\}$), dan definiëren we nu:

$$\begin{aligned} \text{Pr} &= \frac{|R \cap C|}{|C|} \\ \text{Re} &= \frac{|R \cap C|}{|R|} \\ F_1 &= \left(\frac{1}{2}P^{-1} + \frac{1}{2}R^{-1}\right)^{-1} = 2\frac{\text{Pr} \cdot \text{Re}}{\text{Pr} + \text{Re}} \end{aligned} \tag{4.4}$$

In het voorbeeld hierboven zijn er $N = 16$ posities met $\text{Pr} = 2/4$ (d.w.z. 2 van de 4 voorspelde splitsingen is correct) en $\text{Re} = 2/3$ (d.w.z. 2 van de 3 werkelijke splitsingen werd gevonden). Overheen een hele dataset kunnen we een globale precision en recall rapporteren op twee manieren: ofwel door het gemiddelde van alle individuele Pr en Re te nemen (*macro-averaging*), ofwel door de tellers en noemers van Pr en Re te accumuleren en slecht eenmalig te delen, helemaal op het einde (*micro-averaging*). De standaard van Kurimo e.a. (2006) en andere literatuur is micro-averaging, dus gebruik ik die ook.

Foutentaxonomie In het bovenstaande binaireclassificatieprobleem zijn de *positieven* de tussenletterposities met een splitsing in de referentie, en de *negatieven* zijn die zonder. Correct voorspelde splitsingen zijn *echte positieven* (Eng.: *true positives, TP*), correct weggelaten splitsingen zijn *echte negatieven* (Eng.: *true negatives, TN*). Splitsingen die de tokeniser wel plaatst maar niet in de referentie zitten, zijn *valse positieven* (Eng.: *false positives, FP*) en gemiste splitsingen zijn *valse negatieven* (Eng.: *false negatives, FN*). We rangschikken ze in een *verwarringsmatrix* (Eng.: *confusion matrix*):

		kan. +	kan. −
ref.	+	TP	FN
	−	FP	TN

In de literatuur besteedt men geen speciale aandacht aan de kwalitatieve implicatie van valse positieven en valse negatieven. Kurimo e.a. (2006) spreken van "inserties" en "deleties", maar daar blijft het bij. Ik stel voor om beide foutsoorten te bekijken vanuit een framework van vier kwaliteiten:

- *Verklaring*: waarom is het een fout?
- *Schuld*: wat veroorzaakte de fout?
- *Aliasing*: creëert de fout een ander geldig morfeem?
- *Ernst*: behoort de fout tot een cruciale deelverzameling van fouten?

Voor valse negatieven (te weinig gesplitst, te veel gemerged) bestaat de verklaring uit de twee referentiemorfemen die door de missende splitsing gescheiden hadden moeten

zijn. De schuld ligt (in BPE) bij de unieke merge die de spatie tussen beide heeft geconsumeerd. Aliasing treedt op als daardoor een groter morfeem ontstaat. Ernstige valse negatieven zijn die die de twee deelwoorden van een samenstelling doen samensmelten.

Voor valse positieven (te veel gesplitst, te weinig gemerged) is de verklaring het morfeem dat middendoor gesplitst is. De schuld ligt (in BPE) bij elke merge die tijdens de gebruiksfase, die iteratief merges uitvoert, de splitsing had kunnen consumeren en dat niet heeft gedaan. Aliasing treedt op als daardoor minstens één kleiner morfeem ontstaat. Ernstige valse positieven zijn die waarover er geen ambigue referenties bestaan, wat wel zo is voor bv. *vergoeding* (zowel *vergoed|ing* als *ver|goed|ing* zijn acceptabel).

Verschillende secties hieronder behandelen een van die vier kwaliteiten van valse positieven en/of negatieven: §4.14 gaat over positieve en negatieve aliasing, §4.16 gaat over ernstige valse negatieven, en §5.3 bouwt een nieuwe tokeniser a.d.h.v. schuld. Verklaringen traceren laat daarbij toe om foutetokenisatievoorbeelden overvloed te produceren i.p.v. toevallig te ontdekken: Tabel D.4 is volledig automatisch gegenereerd.

Referentiedatasets We hebben ter berekening van Vgl. 4.4 dus een dataset nodig met manuele, morfologisch correcte referentiesplitsingen. Zhou (2018) gebruikt daartoe het corpus van de MorphoChallenge 2010, en Bostrom en Durrett (2020) gebruiken CELEX2. Ikzelf gebruik de lemmata en morfologieën in het e-Lex-lexicon (§C.2) van de Taalunie (2014), waarvan de morfologische referenties overgenomen zijn uit CELEX.[61]

Zoals o.a. Laureys e.a. (2004) en Creutz en Lindén (2004) opmerken, geven CELEX-gebaseerde datasets geen rechtstreekse splitsingen, maar in plaats daarvan een hiërarchie van onderliggende morfemen. e-Lex geeft bv. volgende ontleding voor *isolementspositie*:

```
(((isoleer)[V],(ement)[N|V.])[N],(s)[N|N.N],(((pose)[N],(eer)[V|N.])[V],(itie)[N|V.])[N])[N]
```

Wij zijn geïnteresseerd in een splitsing van de string zelf. Een substring die correspondeert met een onderliggend morfeem (bv. *isol*, gegenereerd door *isoleer*) heet een *morf* (Creutz en Lagus, 2002), analoog aan het verschil tussen een type en een token. Voor een woord van lengte n zijn er meer dan $O(n^n)$ mogelijke aligneringen tussen substrings en de gegeven rij morfemen (zie §B.4), dus enumeratie is niet mogelijk. In §B.4 ontwikkel ik een viterbialgoritme dat voor M morfemen de alignering met maximale karakteroverlap vindt met complexiteit slechts $O(n^2M)$. De uitvoer is bijvoorbeeld

```
(((isol)[V],(ement)[N|V.])[N],(s)[N|N.N],(((pos)[N],()[V|N.])[V],(itie)[N|V.])[N])[N]
```

Zoals ik in §C.2 aangeef, bestaat er naast een dergelijke "morfemische" splitsing ook een "lexemische" splitsing, die verkregen wordt door alle prefices aan het volgende morf te plakken en alle suffices aan het vorige morf. In het voorbeeld van hierboven:

```
((isolement)[N],(s)[N|N.N],(positie)[N])[N]
```

Lexemische splitsing komt van pas om te focussen op díé morfemische splitsingen die de deelwoorden van een samenstelling scheiden, met name in §4.16.

Gewichtsdatasets Een lexicon bevat geen duplicaten; elk lemma komt eenmalig voor. In lopende tekst, d.w.z. tokens i.p.v. types, volgen lemmata zoals gewoonlijk een zipfiaanse verdeling. Bij micro-averaging van Vgl. 4.4 kunnen de teller en de noemer van elk lemma w ofwel exact één keer meetellen, ofwel dupliceren we ze $C_{\mathcal{D}}(w)$ keer en is

[61] Via WebCelex kan iedereen met een universitaire account inhoud ophalen uit CELEX, en verifiëren dat hij met e-Lex overeenkomt: Create Lexicon » Dutch Lemmas » StrucLab » Ok » Ok » Include Word » Ok.

het dus erger om een veelvoorkomend woord fout te segmenteren. Eender welk corpus $\mathcal{D}$ kan die tokengewichten aanleveren; Bostrom en Durrett (2020) gebruiken bv. de Engelse Wikipedia. Ik gebruik OSCAR-nl, waarbij $C_{\mathcal{D}}(w) \equiv 1$ indien $w \notin W_{\mathcal{D}}$.

Resultaten Tabel 4.2 vergelijkt BPE_{nl} met de BPE-tokenisers waarvoor Zhou (2018) en Bostrom en Durrett (2020) resultaten publiceren. Recall en F_1 liggen systematisch onder de 20%. Bemerk dat de Nederlandse precision het hoogst is en de recall het laagst. Aannemelijk komt dat doordat het vocabularium van BPE_{nl} het grootst is en dus veel woorden memoriseert i.p.v. splitst.

Auteur	Tabel	Taal	Referentiecorpus	$\lvert V\rvert$	Pr	Re	F_1
Zhou (2018)[62]	3.3	Eng.	MorphoChallenge 2010	32k	0.168	0.199	0.182
Bostrom e.a. (2020)[63]	3	Eng.	CELEX2 (tokens in Wikipedia)	20k	0.386	0.129	0.193
(ik)	4.2	Ned.	e-Lex (tokens in OSCAR)	40k	0.543	0.115	0.190
			e-Lex (types)		0.522	0.550	0.536

Tabel 4.2 – Overeenkomst van BPE-splitsingspunten met morfologische splitsingspunten.

4.14 Aliasing (Ataman e.a., 2017)

Samengevat – In de letters van het ene morfeem kunnen die van een ongerelateerd morfeem vervat zitten. BPE herkent niet wanneer welk morfeem bedoeld wordt.

In de literatuur Tabel 1.1 geeft twee interessante BPE-tokenisatiefouten die Ataman e.a. (2017) in hun Turks→Engels-systeem waarnemen. Ten eerste: het woord *kanunda* komt van *kanun* ("de wet") en de suffix *-da* ("in").[64] BPE segmenteert het foutief als *kan|unda*. *-unda* is een afvaltoken dat de NMT-encoder benaderend als *-da* codeert. Cruciaal is echter dat *kan* ("het bloed") een bestaande woordstam is die het model ook herkent; in totaal benadert het *kanunda* dus als *kanda* ("in het bloed"). Uit de tokenisatie *kan|unda* kunnen we afleiden dat áls het type *kanun* bestaat in de BPE-tokeniser, zijn merges minder prioritair zijn dan die van *unda*. Mogelijks is de prioriteit zo veel lager dat andere uitgangen dan *-da* er eveneens in slagen om *un* naar zich toe te trekken vooraleer *kan* dat kan doen. We leiden ook af dat *kanunda* zelf geen type is, want anders – zie §4.17 voor een formeel bewijs – bestond de merge *kan+unda* die de splitsing zou redden.

In hun tweede voorbeeld gebeurt exact hetzelfde: *ağlama|yacak*, afkomstig van *ağlamak* ("huilen"), wordt door hun BPE-tokeniser gesplitst als *ağ|lamayacak*, waarbij de betekenis van *ağ* ("net") de vertaling beïnvloedt. BPE zit met een fundamenteel probleem in dit voorbeeld: voor vervoegingen van "huilen" met stam *ağlama* moet BPE blijven mergen tot het één token is, terwijl bij verbuigingen van "net" met stam *ağ*, in het meervoud bv. *ağlar*, er zo weinig mogelijk gemerged mag worden opdat *ağ* en *la* niet samentrekken. BPE kan die nuance niet maken: het is het een of het ander.

Tweemaal is het probleem dat in het ene morfeem een ander morfeem – een *alias* – ontdekt wordt. In signaalverwerking zorgt *aliasing* voor informatieverlies. In NMT treedt

[62] *Opgelet:* Zhou telt precision en recall lichtjes anders, nl. op basis van overeenkomstige subwoorden i.p.v. splitsingen. Stel dat een woord *abc* te splitsen is als twee morfemen *ab|c* maar de tokeniser *a|b|c* aangeeft, dan is de precision van de subwoorden 1/3, terwijl de precision van de splitsingen 1/2 is.

[63] *Opgelet:* Bostrom en Durrett geven aan dat hun alfabet 4000 karakters groot is, waardoor ze veel minder dan 20 000 merges leren.

[64] De aandachtige lezer herkent die suffix ook in de bespreking van klinkerharmonie in §4.4.

er eerder kennisverlies op achteraf: Ataman e.a. (2017) tonen (in hun tabel 5) dat BPE de woordstam in *ağ|lar|ın* ("hun netten") wel degelijk correct terugvindt en de tokens *ağ|ların* produceert, maar het NMT-systeem *alsnog* de foute vertaling "huilen" produceert. Dat is perfect verklaarbaar: in voorbeelden met "huilen" (*ağlama*) heeft de tokeniser genoeg keren foutief het token voor "net" (*ağ*) aan de encoder gegeven dat het systeem *ağ* is beginnen te associëren met "huilen". Er is een soort *afstandseffect* waarbij incorrecte segmentaties vanop afstand de correcte segmentaties nutteloos maken.

Huck e.a. (2017) schrijven dat ze in hun prefixsplitser (cfr. §2.5.3) speciale regels moesten inbouwen omdat zelfs prefices elkaar aliasen. Ter illustratie: *onmogelijk* is het tegengestelde van *mogelijk* (prefix "on"), maar een *onderzeeër* is niet het tegengestelde van een "*derzeeër*" (prefix "onder"). De reden waarom ze de prefixsplitser uit CSCS-BPE weglaten, is weliswaar niet aliasing, maar lexicalisatie: zo is iemand *onmenselijk* het tegengestelde van *menselijk*, maar een *onmens* is wel degelijk een mens.

In e-Lex Zoals Tabel C.3 toont, zijn er 13 317 unieke morfemen in e-Lex die aan de oppervlakte komen als 18 516 unieke morfen die elkaar kunnen aliasen. De bovenlimiet op het aantal aliasende morfparen is $\binom{18\,516}{2} = 171\,430\,386$. De meeste daarvan zullen niet aliasen, en bovendien zijn sommige morfparen afkomstig van hetzelfde morfeem en dus niet aliasend. Anderzijds zijn alle letters van het alfabet een morf in e-Lex; de letters van een morf aliasen het morf dus, maar dergelijke gevallen zijn niet nuttig om te bestuderen. We moeten ook niet per se elke geldige alias beschouwen om het probleem te illustreren; dat zou ook overbodig veel rekenwerk vragen. Vandaar beperk ik mij tot *prefixaliasing*: morfen die beginnen met kleinere morfen, waarbij dus geen aliases horen die alleen in het midden of op het einde van een morf voorkomen.

Daartoe stel ik een prefixboom op van de 13 317 morfen in e-Lex, hetgeen slechts lineaire tijd kost in het aantal morfen. Het zoeken naar prefixaliases is nu triviaal: elk paar knopen op elk pad van een blad naar de wortel van de prefixboom is een aliasend paar. Er zijn zo in totaal 24 811 unieke paren, geteld vanaf diepte 3 (d.w.z. zonder morfen van lengte 1, noch de wortel daar die de lege string bevat). Noem die morfen op diepte 3 de "stamouders". Figuur D.15 toont een deel van de prefixboom onder de stamouder *po*, die het langste pad van blad tot stamouder heeft (nl. 7 knopen). Merk op dat de bladeren van de boom vaak volledige woorden zijn; dat is deels idiopathisch in e-Lex, en deels omdat sommige lange woorden (bv. *polaroid*) effectief gewoon hun eigen morfeem zijn. Er is niet echt een vaste diepte waarop morfen groot genoeg maar niet te groot zijn, dus ik filter verder geen morfen.

Herinner de definities van aliasing door valse positieven en valse negatieven in §4.13. Een morf m in de prefixboom heeft als valsnegatieve aliases al zijn afstammelingen, omdat die kunnen verschijnen door m te absorberen in een te weinig gesplitst token. Omgekeerd heeft m als valspositieve aliases al zijn voorouders, omdat die kunnen verschijnen door m te splitsen. Elk voorkomen van m loopt het gevaar om positief gealiast te worden. Op negatieve aliases is er alleen gevaar wanneer m samen met de letters van minstens één negatieve alias voorkomt.
Bijvoorbeeld: *bes* is een positieve alias van *besnaard* waardoor die laatste altijd gevaar loopt op positieve aliasing (bv. in *fijnbesnaardheid*). Daarentegen is *besnaard* een negatieve alias van *bes* en vormt niet altijd een gevaar (bv. in *besjesgelei*, omdat de letters "*naard*" er niet achter staan). In *accommodatieflat* is er dan weer wel gevaar op negatieve aliasing, omdat dat het morf *atie* bevat met als een van zijn negatieve aliases *atief* (als in *commutatief*). BPE_{nl} segmenteert *accommodatieflat* effectief als *accommod|atief|lat*, waarin overigens ook positieve aliasing plaatsvindt tussen *flat* en *lat*. Dat laatste detecteer ik niet in dit experiment omdat *lat* een suffixalias en geen prefixalias is.

	morfen				morfemen			
	gevaren	$\%_{\textbf{tot}}$	**effectief**	$\%_{\textbf{gev}}$	**gevaren**	$\%_{\textbf{tot}}$	**effectief**	$\%_{\textbf{gev}}$
FP	147 622	61.85%	16 015	10.85%	123 743	51.7%	11 642	9.41%
FN	16 860	7.06%	5787	34.32%	10 077	4.21%	2903	28.81%

Tabel 4.3 – Positieve en negatieve prefixaliasing in e-Lex. "Gevaren" is hoeveel morfen resp. morfemen in e-Lex-lemmata gevaar lopen op minstens één geval van aliasing, en "effectief" is hoeveel van die gevaren materialiseren bij gebruik van BPE_{nl}. Voor de linkerhelft zijn 238 678 morfen beschouwd, voor de rechterhelft 239 357 morfemen.

Gegeven de aliases van elk morf, ga ik nu als volgt te werk: voor elk lemma in e-Lex genereer ik de lijst van morfen, en voor elk morf check ik of er een gevaar is op aliasing (heeft het aliases en zijn de correcte letters aanwezig in het lemma). Waar er gevaar is, check ik of de gevaarlijke alias aanwezig is als token in de segmentatie met BPE_{nl}, al dan niet met SOW. Tabel 4.3 toont de resultaten voor zowel positieve en negatieve prefixaliasing. De linkerhelft toont het experiment zoals hierboven besproken, en de rechterhelft herhaalt het maar met een prefixboom van morfemen i.p.v. morfen.

De tabel is zoals verwacht: de rechterhelft heeft op alle vlakken lagere getallen omdat morfemen vaak niet in hun geheel voorkomen en dus ook niet in tokens te vinden zijn. In de linkerhelft zien we dat er gevaar op positieve aliasing is in meer dan de helft van alle tegengekomen morfen, waarvan er "slechts" 10% materialiseert (à la *kan|unda*). Gevaar op negatieve aliasing is veel zeldzamer vanwege de voorwaarde die ik hierboven beschreef, maar daardoor wordt 1 op de 3 gevaren ook werkelijkheid (à la *accommod|atief|lat*).

4.15 Typofragiliteit (Provilkov e.a., 2020)

> **Samengevat** – Spelfouten kunnen de BPE-segmentatie van een woord sterk veranderen.

De benchmarks waarop LM's gefinetuned worden (Devlin e.a., 2019) bestaan traditioneel uit manueel geselecteerde (of gegenereerde) zinnen zonder taalfouten. In industriële toepassingen is de invoer van een LM minder geïdealiseerd, gezien die vaak van een gebruiker komt: men kan een LM bv. op sociale media loslaten (Nguyen e.a., 2020), waar het te maken krijgt met spelfouten, verkortingen, emoji's, accountnamen en hashtags. Sterker, een taalmodel kan ingezet worden als geavanceerde spelling- en grammaticachecker in onderwijstools zoals de ILT-schrijfhulp waar het juist expliciet de bedoeling is dat het LM foutieve taal tegenkomt.

BPE-segmentatie verloopt hiërarchisch, incrementeel en greedy: elke iteratie evalueert alleen de merges van aangrenzende tokens resulterend uit de vorige iteratie. Stel dat er zo drie tokens $t_\ell\, t\, t_r$ zijn; een merge van t met de linkerbuur t_ℓ impliceert dat de merge tussen t en de rechterbuur t_r nooit meer beschikbaar is, en als $(t_\ell t, t_r) \notin M$ kan die verbinding alleen nog gemaakt worden indien de strings links van $t_\ell t$ en rechts van t_r grotere subwoorden maken die uiteindelijk wel nog mergen. Een merge met de ene buur heeft dus de mogelijkheid om een permanente splitsing met de andere buur te vormen.

Spelfouten (*typo*'s) zorgen voor onverwachte buren en dus het onverwachts wegvallen van correcte merges, wat zich kan propageren doorheen de ganse string. Ik test die hypothese door het aantal tokens te tellen in (unieke) woorden voor en na het invoegen van typo's; om computatietijd te verlagen beperk ik me tot typo's met *levenshteinafstand* gelijk aan 1 (d.w.z. 1 karakter mag worden verwijderd, gesubstitueerd, of ingevoegd), en ook

karakterwisselingen (een deelverzameling van alle typo's met 2 substituties). Daartoe neem ik de lemmata in e-Lex en ook de lemmata in het NT2Lex-lexicon van Tack e.a. (2018), en doe ik voor de vier soorten typo een gelijkaardig experiment: per lemma kies ik een willekeurige positie om de typo op te plaatsen, en voor insertie en substitutie ook een willekeurige kleine letter (a t.e.m. z). Ik tokeniseer het oorspronkelijke woord en het bewerkte woord beide met BPE_{nl}.

Ik meet drie aspecten: het aantal tokens in elk van de twee (marginale verdelingen), het verschil in die aantallen (gepaarde verdelingen) en ten slotte ook de precision en recall van splitsingspunten (Vgl. 4.4) voor en na introductie van de typo. Ingeval er een invoeging of verwijdering gebeurt, aligneer ik de strings op de intuïtieve manier; stel dat voor een string *abcd* de letter *b* verwijderd wordt, dan is de alignering van splitsingspunten

Origineel: `a | b | c | d`
Verwijdering: `a |   c | d`

...waarbij elk van de verticale splitsingen aanwezig of afwezig is o.b.v. de tokenisatie. Tabel 4.4 toont de resulterende precision en recall van de experimenten, en de histogrammen van tokenaantallen (zestien in totaal) zijn te vinden in Appendix D (§D.8).

Uit de tabel blijkt dat na elk soort typo zo'n 80% van de oorspronkelijke splitsingen in de dataset worden teruggevonden, maar tegelijk is amper de helft van alle nieuwe splitsingen ook in de oorspronkelijke te vinden. De histogrammen tonen inderdaad veel dikkere staarten richting hoge tokenaantallen. Uit de histogrammen zijn er nog twee conclusies te trekken: enerzijds zorgt elk van de vier soorten typo's voor een verhoogd aantal tokens, equivalent met een verlaagd aantal merges of een verhoogd aantal tegengehouden merges. Anderzijds zijn er gemiddeld meer tokens naarmate de woorden met typo's meer karakters hebben: deleties hebben de minste tokens, dan substituties, dan verwisselingen (dubbele substituties), en dan inserties. Dat houdt steek: meer karakters hebben meer tokens nodig. Let wel dat karakters alleen het aantal tokens niet volledig verklaren, gezien de woorden zonder typo's *meer* karakters bevatten dan de woorden met deleties en toch *minder* tokens.

	deletie		**substitutie**		**wisseling**		**insertie**	
	e-Lex	NT2Lex	e-Lex	NT2Lex	e-Lex	NT2Lex	e-Lex	NT2Lex
Pr	0.58	0.38	0.50	0.32	0.44	0.28	0.46	0.30
Re	0.79	0.78	0.87	0.87	0.80	0.80	0.89	0.89

Tabel 4.4 – Precision en recall van de splitsingspunten die de BPE_{nl}-tokeniser ingelast in woorden met typo's t.o.v. zonder.

Om een LM met typo's te laten omgaan, zijn er mijns inziens vier tokeniseraanpassingen mogelijk. Ten eerste kan men de tokeniser laten voorafgaan door een spelchecker, waarbij de technologie in § 2.4.1 van pas komt. Ten tweede kan men $|V|$ verlagen, waardoor er minder grote merges bestaan en een typo dus minder merges kan blokkeren dan hierboven. Ten derde kan men het LM regulariseren met probabilistische segmentaties; Provilkov e.a. (2020) toont zo dat BPE-dropout voor NMT-modellen zorgt die – na te trainen op een corpus zonder typo's – beter presteren op corpora met typo's in de invoerwoorden. Een laatste oplossing lijkt me om een viterbi-gebaseerde tokeniser te gebruiken, aangezien die tracht om globaal gezien een woord zo goed mogelijk te segmenteren, en kleine foutjes dus door de vingers ziet.

4.16 Samenstellingsgrensoverschrijdend gedrag

Samengevat – BPE ziet alle aangrenzende karakters als bewijs dat ze samenhoren, dus merget het vaak de overgang tussen de karakters van twee samengeplaatste lemmata.

4.16.1 Lemmagrenzen

Samenstellingen zijn best niet te negeren: Berton e.a. (1996) ondervinden dat in de top T frequentste woordtypes van hun Duits corpus het percentage aan samenstellingen 60% overstijgt bij toenemende T, terwijl het percentage aan samenstellingen in hun woordtokens convergeert tot 20%. Fritzinger en Fraser (2010) vinden dezelfde 20%.

Samenstellingen in het trainingcorpus zorgen voor een vertekend beeld van de BPE-metriek, gezien de overgang tussen de deelwoorden (lemmata, lid van twee aparte lexemen) meetelt bij de rest van de frequenties (bv. de *rt* in *master<u>t</u>hesis*). Het probleem is dat BPE aanneemt dat letters (of algemener, tokens) aaneengeplakt mogen worden wanneer ze naast elkaar staan. In een samenstelling is de overgang tussen de twee deelwoorden juist *minder* bewijs dat de tokens aan weerszijden samenhoren, niet *meer*! Om consistent te zijn met morfologie zouden dergelijke overgangen op z'n minst *niet* mogen meetellen in lijn 7, en het is zelfs een idee om ze *negatief* mee te tellen.

Het idee van Huck e.a. (2017) om eerst de grenzen van prefices, suffices en lemmata af te bakenen met spaties, is niet voldoende om een BPE-tokeniser te leren die die grenzen respecteert. Stel dat we dat doen, dan zorgt het populaire morfeeem *-ing* nog steeds voor hoogprioritaire merges *i+n* en *in+g*, waardoor zo'n tokeniser nog stéédsde woordgrens foutief samensmelt in een samenstelling als *zingeving*. Tabel D.3 toont de eerste merges van een BPE-tokeniser die op die manier getraind is, en inderdaad: de merges *i+n* en *in+g* behoren tot de eerste tien merges.
Of de tokeniser nu geleerd wordt over een corpus van woorden of een corpus van morfemen, er lijken steeds bepaalde letters/tokens te zijn die aan het begin of einde van woorden voorkomen en systematisch de neiging hebben om deel te nemen aan een merge in de foute richting. Hieronder bespreek ik welke rol interfices daarin spelen, maar het probleem is breder dan dat: Tabel D.4 toont dat de laatste *a* in *propaganda* een enorm sterke affiniteit heeft voor de eerste letter van eender welk woord dat er rechts mee wordt samengesteld, analoog aan de laatste *a* in *corona*, cfr. Figuur 1.2. De rechtswaartse affiniteit van de letter *a* wordt ook duidelijk in Figuur D.18, een kaart van de gemergede letters in de eerste 200 BPE_{nl}-merges. In §5.3 probeer ik pathologische affiniteiten aan te kaarten.

In die twee voorbeelden is het probleem van samengetrokken lemmata trouwens niet gewoon een kwestie van het samentrekken van die lemmata. Het is veel erger dan dat: de lemmagrens in *zingeving* wordt niet overbrugd door een merge *zin+geving*, maar wel *in+g*. Daardoor is de oplossing niet om τ te verlagen om te vermijden dat types te groot worden. De allereerste merge die BPE_{nl} uitvoert is *e+n*. Dat betekent dat *reeds na de allereerste BPE-iteratie* samenstellingen als *familie<u>n</u>aam*, *zee<u>n</u>iveau*, *injectie<u>n</u>aald* ... een samengesmolten lemmagrens hebben en hun deelwoorden onzichtbaar worden.[65] Figuur D.22a toont de verdeling van de merges die voor meer dan 50% van hun toepassingen zorgen voor een samentrekking van een lemmagrens (extreem pathologische merges, dus). We zien effectief dat sommige van de allervroegste merges al zo extreem de mist in gaan. Meer over de implicaties daarvan in §4.17.

[65] De populaire merge *d+e* ligt bv. aan de basis van tokenisaties als *Ġb|ade|end|je* en *Ġt|ande|x|tractie*.

Om te beoordelen hoezeer een tokeniser de lemmagrenzen respecteert, kunnen we splitsingsposities vergelijken net als in §4.13. De referentie is nog steeds gebaseerd op morfologische annotaties in e-Lex, alleen zijn nu sommige van de *morfemische* splitsingspunten verwijderd opdat er voor elk segment een lexeem bestaat. De exacte constructie van die *lexemische* splitsingspunten staat uitgelegd in §C.2.
De problematiek hierboven gaat voornamelijk over recall, nl. hoeveel van de lemmagrenzen in de referentie de tokeniser behoudt. Precision meet in dit geval welke fractie van alle voorspelde splitsingen lexemisch zijn, en is minder interessant aangezien we het juist goed vinden als de tokeniser bv. suffices afsplitst. Aangezien de lexemische splitsingpunten een deelverzameling van alle morfemische splitsingspunten zijn, moet de precision sowieso lager zijn dan in Tabel 4.2 (minder echte positieven, zelfde voorspellingen). De recall kan beide kanten uit (minder echte positieven, minder referenties).

Tabel 4.5 toont dat BPE_{nl} lemmagrenzen deftig respecteert overheen een brede woordenschat (30% foutief samengetrokken), maar dat de gemaakte fouten kennelijk in veelvoorkomende woorden zitten (30% correct behouden).

	Pr	**Re**	F_1
types	0.39	0.70	0.50
tokens	0.36	0.31	0.33

Tabel 4.5 – Overeenkomst tussen BPE_{nl}-splitsingspunten ($|V| = 40$k getraind op OSCAR-nl) en lexemische splitsingspunten voor lemmata in e-Lex. De rij "tokens" is hetzelfde resultaat maar gewogen met de frequentie in OSCAR (of 1 indien afwezig, cfr. Tabel 4.2).

4.16.2 Interfices

De onvoorspelbaarheid van interfices werd reeds in § 1.3.2 geïllustreerd. Een interfix zorgt ervoor dat een samenstelling soms meer dan de concatenatie van twee lemmata is. In §2.4.1 kwam dan ook aan bod hoe de systemen van Koehn en Knight (2003), Stymne (2008) en Macken en Tezcan (2018) proberen om een potentiële interfix te omzeilen. Voor de BPE-tokeniser zijn er twee nadelen: de ene tijdens vocabularisatie, de andere tijdens segmentatie. Ik beperk mij hieronder tot een analyse van de tussen-*s*, die 62.5% van alle interfices in e-Lex bedraagt. (De andere interfices zijn 25% *en* en 12% *e*.)

Woorden die regelmatig links of rechts van een interfix-*s* voorkomen, riskeren om door BPE gezien te worden alsof ze eindigen resp. beginnen met een *s*. Dat zorgt voor inflatie van het subwoordvocabularium, gezien eenzelfde type een kopie met *s* vooraan en een kopie met *s* achteraan kan krijgen. Herinner bovendien dat een model niet zomaar toegang heeft tot de karakters van tokens (cfr. Figuur 1.1), en de relatie tussen die kopieën dus niet evident is.

Natuurlijk kunnen morfemen en woorden ook gewoon met *s* beginnen of eindigen, dus kunnen we ter bewijs van die inflatie niet zomaar op zoek gaan naar subwoorden in V die zowel met als zonder begin-/eind-*s* bestaan. Het kan echter directer: eerst verzamelen we alle e-Lex-lemmata met een interfix-*s* in de morfologie, en vervolgens laten we BPE_{nl} die lemmata tokeniseren. Als er redundante interfixtypes in het vocabularium zitten, dan zullen die ervoor zorgen dat de *s* geen apart token krijgt.
Tabel 4.6 bevestigt dat: van de 9622 lemmata met interfix-*s* in e-Lex is die interfix in 95% van de tokenisaties gefuseerd met de letters links, rechts, of zelfs volledig opgenomen in een groter token. De *s* heeft duidelijk een grote affiniteit voor linkswaartse merges, bv. in de populaire merge *ing+s*. De verwarring van de tokeniser wordt bemoeilijkt door het feit dat 7.17% van alle morfemen in e-Lex wel degelijk eindigt met een *s*.

	Aantal	Percentage
Geen	514	5.34%
Alleen links	6074	63.13%
Alleen rechts	2655	27.59%
Beide	379	3.94%

Tabel 4.6 – Merge-affiniteiten van elke *s* die e-Lex als interfix aangeeft.

De aanwezigheid van interfices in de trainingdata zorgt nadien voor gevaarlijke verwarring tijdens segmentatie. Enerzijds zijn er type I-fouten: gevallen waarin BPE denkt een interfix te hebben gevonden en die letter naar links merget, terwijl het eigenlijk de eerste letter van de samenstellingskern was. *belastingsysteem* wordt op die manier *belast|ings|ysteem*. Het toevoegen van een *s* achteraan een woord zorgt in het Nederlands gelukkig meestal niet voor een lexeemverandering. De situatie is veel erger voor type II-fouten: gevallen waarin BPE een aanwezige interfix niet herkent en naar rechts merget. Dat kan namelijk in een bestaand kernlexeem resulteren, zoals gebeurt in *staat|smacht*, *ambtenaar|sloop|baan*, *af|god|stempel*, *volk|stelling*, *wijsheid|stand* en *handel|staal*. In e-Lex zijn er in totaal 246 lemmata waarin het gevaar op een dergelijke verwarring bestaat, en in 156 daarvan (63%) gebeurt ze ook.

4.16.3 Starttypes als einde

In het Nederlands staat de *samenstellingskern* achteraan: men kan *tafelbier* op een *biertafel* drinken, omdat tafelbier bier is en een biertafel een tafel, niet omgekeerd. Een *dodehoekongeval* is geen dode en geen hoek, maar een ongeval.

Welnu, stel dat alle lemmagrenzen in samenstellingen gerespecteerd worden – een idealisering, cfr. Tabel 4.5 – dan is het doel van subwoordtokenisatie dat we reeds opgedane kennis i.v.m. de kern kunnen gebruiken bij de interpretatie van de volledige samenstelling: alles wat het model weet over een ongeval zou hergebruikt moeten worden voor een dodehoekongeval, en dus is het uiteraard beter om de string *ongeval* in beide woorden dezelfde tokenisatie te geven.

Jammer genoeg gebeurt dat niet in BPE_{nl}. De oorzaak is het gebruik van SOW i.p.v. EOW (cfr. Tabel 4.1): de tokeniser kán niet dezelfde tokens produceren voor een losse kern en een gebonden kern, want de losse kern begint verplicht met een SOW terwijl de gebonden kern nooit in het begin van de samenstelling staat en dus nooit een SOW krijgt.

	na laatste merge	voor laatste merge
referentie	Ġdode|hoek|on|geval	...
goed	Ġ|ongeval	...
matig	Ġongeval	Ġ|ongeval
slecht	Ġongeval	Ġongev|al
	Ġong|ev|al	...

Tabel 4.7 – Mogelijke verschillen in tokenisatie van een losse en een gebonden samenstellingskern.

Hoe erg is het verschil? In het beste geval bevat de losse kern exact dezelfde tokens als de gebonden kern, en komt er alleen een los SOW-token bij. Minder optimaal, maar remedieerbaar met BPE-dropout, is wanneer dat losse SOW-token met de allerlaatste merge aan de kern gebonden wordt. In het ergste geval is het SOW-karakter niet als

laatste gemerged, en is de tokenisatie dus sterk verschillend, zelfs met dropout. Tabel 4.7 geeft een voorbeeld van de drie situaties. De statistieken zijn niet gunstig: 66% van alle types in BPE_{nl} (26 429 van de 40 000) hebben een SOW-karakter, waarvan slechts een kwart (6570 van de 26 429) ook zonder SOW-karakter voorkomt. Met andere woorden: zo'n 50% van het vocabularium (19 859 types) wordt noodgedwongen significant anders getokeniseerd indien gebruikt als samenstellingskern i.p.v. alleenstaand.

4.17 Functionele erfzonde

Samengevat – BPE kan nooit twee mergeregels leren die in hetzelfde type resulteren. Bovendien vinden merges permanent en in het hele corpus plaats.

De idee van een *erfzonde* (Eng.: *original sin*) is dat het begaan van één misstap in het verleden permanente negatieve gevolgen nalaat in de toekomst. De biologie kent zo ook het fenomeen van *original antigenic sin*: het immuunsysteem leert tijdens de eerste infectie met een nieuw virus welke proteïnen het virus herkenbaar maken (de antigenen), en leert antilichamen produceren om daaraan te binden en zo de infectie te genezen. De geheugencellen met kennis van die eerste infectie zullen tijdens een tweede infectie domineren, zelfs als het virus zo geëvolueerd is dat het beter zou zijn om betere antilichamen te leren aanmaken. Daardoor duurt het genezen langer. De eerste impressie van de virale antigenen is dus bepalend voor al dan niet permanente negatieve gevolgen.

4.17.1 Concept en bewijs

Figuur 1.2 toont dat de BPE-tokeniser van RobBERT geleerd heeft om het subwoord *coronamaatregelen* op te bouwen uit de subwoorden *coron* en *amaatregelen*. We hadden het liever anders gezien, nl. de concatenatie van *corona* en *maatregelen*. Het probleem is nu dat BPE lijdt aan wat ik de *functionele erfzonde* (Eng.: *original functional sin, OFS*) noem: eens een subwoord gevormd is door een merge, is het onmogelijk om een alternatieve merge te leren die hetzelfde subwoord vormt. Anders gezegd is elk subwoord (als het niet in het alfabet zat) het resultaat van exact één merge in M, wat een bijectieve functie $V \to M$ definieert. Nog anders gezegd heeft elk subwoord buiten het alfabet exact twee ouders in de mergegraaf, nl. exact één linker- en exact één rechterouder,[66] beide weer een functie $V \to V$.

Bewijs: Het bewijs voor OFS volgt uit de leerfase in Algoritme 2.1 en volgende eigenschap: *na de iteratie waarin lijn 7 een paar x_y kiest en lijn 8 het vervangt door xy, bestaat er nergens in het trainingcorpus nog een sequentie met meer dan 1 token waarvan de concatenatie xy is.* Beschouw alle voorkomens van de string xy (bv. $x =$ *coron* en $y =$ *amaatregelen*) in het trainingcorpus, dat reeds in woorden is opgesplitst met een pretokeniser. Er zijn twee gevallen: xy als woord apart, en woorden $wxyz$ (bv. *coronamaatregelenprotest*, *ex-coronamaatregelen*, *anti-coronamaatregelenbeweging* ...) waarbij w en z strings zijn met netto minstens één karakter ($|wz| > 0$). Indien er vóór de merge tot xy een merge plaatsvindt waarbij er in zo'n woord $wxyz$ karakters samenklitten van w en x of van y en z, dan is het onmogelijk om met verdere merges ooit nog het subwoord xy in dat woord te verkrijgen. Voor alle woorden $wxyz$ waarin die situatie zich niet voordoet, kunnen we w en z negeren totdat xy tot één subwoord gemerged is, waardoor de substring xy zich gedraagt alsof het de alleenstaande string xy is. Aangezien elke toepassing van lijn 8 hetzelfde effect heeft op alle voorkomens van de alleenstaande string xy, en alle andere voorkomens van xy zich ofwel alleenstaand

[66] In de graaf van Figuur 1.2 arriveren er bijvoorbeeld altijd exact nul of twee pijlen in elke vertex, en vertrekken er onbepaald veel uit elke vertex.

gedragen ofwel nooit tot het subwoord xy kunnen worden gemerged, is er na de merge van x_y tot het subwoord xy geen enkel woord meer waarin na verloop van tijd opnieuw xy wordt gevormd. □

4.17.2 Implicaties van OFS

Algemeen De twee kerngedachten van OFS zijn 1. dat elk subwoordtype s slechts op één manier gemerged kan worden door eenzelfde BPE-tokeniser, aangezien 2. de merge overal tegelijk in het corpus wordt uitgevoerd, en er geen enkele sequentie van tokens meer in het corpus achterblijft die na concatenatie opnieuw s vormt. Door dat laatste is er immers geen data meer die BPE kan informeren over een alternatieve merge voor s, en zelfs als we zo'n merge aan de mergelijst zouden toevoegen, dan nog zou hij niet gebruikt worden vanwege het effect dat ik in §4.9 beschreef i.v.m. RobBERT-2022.

Merk op dat het probleem van OFS *niet* hetzelfde is als determinisme zoals besproken in §4.5: er kan namelijk perfect een deterministische tokeniser bestaan die niet lijdt aan de effecten hierboven, nl. de ideale tokeniser die elk woord altijd op dezelfde correcte manier segmenteert. Het deterministische aspect van OFS is dat een merge overal waar mogelijk wordt uitgevoerd, maar dat is niet hetzelfde.

OFS is ook niet hetzelfde als de aanwezigheid van afval zoals gezien in §4.10: immers, neem eender welke samenstelling die BPE correct splitst, dan is het geen probleem dat er afvaltypes gebruikt zijn om de brug te bouwen van karakters naar deelwoorden (zie bv. de boom die resulteert in het token *master* in de rechterhelft van Figuur D.7).

Samenstellingen We zagen in §4.16 dat het slechts één merge kost om de lemmagrens van een samenstelling foutief te overbruggen, en dat dat reeds plaatsvond voor merges van twee karakters. De implicatie voor de deelwoorden volgt uit het bewijs van OFS: neem dit keer één van de deelwoorden als xy, en stel de hele samenstelling voor als $wxyz$ met $|wz| > 0$. Indien de grens van de deelwoorden overschreden is, werd in het bewijs aangehaald dat het *onmogelijk* is om met verdere merges ooit nog het deelwoord xy als token te verkrijgen in de samenstelling $wxyz$, noch tijdens de leerfase van BPE, noch tijdens de gebruiksfase.

Er is dus weer sprake van dataverlies. Elk voorkomen van een woord in een samenstelling waarin de lemmagrens overbrugd is, is niet langer informatief over dat woord, omdat het begin of einde van het woord is opgegaan in een token dat bijkomende karakters bevat. De samenstellingen in Tabel D.4 zijn een mooi voorbeeld: het is duidelijk dat het token *Ġpropaganda*, dat wel degelijk bestaat (bovenaan de tabel), nergens nog kan ontstaan, zelfs niet als BPE de merge *Ġpropag+and* nog bijleert. Dat wil dus zeggen dat het trainingcorpus alle samenstellingen in de tabel verliest als contexten voor het type *Ġpropaganda*, terwijl een menselijke lezer die wel gewoon kan gebruiken. Bovendien lijdt BPE zelf ook dataverlies: de uiteindelijke merge die *Ġpropaganda* construeert is *Ġpropag+anda*. Vermits de laatste *a* in alle samenstellingen van Tabel D.4 rechtswaarts bewogen is, kan BPE die merge alleen maar afleiden uit de losse strings *propaganda* in het corpus, zonder dat de samenstellingen in Tabel D.4 aan de frequentie bijdragen.

Verlies van contexten is uiteraard nadelig vanwege de DH, maar in dit geval gaat het ook in tegen de filosofie van subwoordtokenisatie in §1.3.3. Daar zagen we dat het Frans t.o.v. het Engels onderbedeeld wordt door woordtokenisers, vermits het Frans sterk flexionele vervoegingen kent en de kennis van één lexeem daardoor verspreid ligt overheen veel meer woordtypes. De bedoeling van subwoordtokenisatie is om die onderbedeling tegen te gaan, maar nu blijkt dus dat het Nederlands op zijn beurt weer onderbedeeld wordt

t.o.v. het Frans of het Engels: in die talen is de bovenstaande samenstelling respectievelijk *campagne de propaganda* en *propaganda campaign*, wat betekent dat *propaganda* dezelfde segmentatie heeft binnen en buiten samenstellingen. In het Nederlands zorgt BPE reeds na enkele iteraties dat er wél een onderscheid is tussen de twee, en ze dus niet langer contexten geven aan dezelfde tokens.

Er is geen weg terug eens de lemmagrens overbrugd is. Daar beide deelwoorden in dat geval eindigen resp. beginnen met een token dat bij losse tokenisatie nooit voorkomt (bv. *propaganda* kan als laatste token nooit *ac* hebben, noch kan *campagne* het als eerste token hebben), ziet hun tokenisatie er mogelijks helemaal anders uit. De situatie is vergelijkbaar met de introductie van een typo (§4.15).

BBPE De problemen die zich voordoen met permanente merges overheen lemmagrenzen doet zich ook voor op byteschaal in BBPE. De coderingen van alle niet-ASCII-codepoints bestaan in UTF-8 uit groepjes van meer dan één byte. UTF-8 is prefixvrij, wat betekent dat eender welke segmentatie in andere groepjes minstens één nonsensgroepje bevat. Het is dus belangrijk om de grenzen van de codepoints te respecteren.

Welnu, stel dat we drie exotische karakters $\mathcal{A}$, $\mathcal{B}$ en $\mathcal{C}$ hebben (bv. drie letters in een van de alfabetten in Tabel C.1), waarbij de strings $\mathcal{AB}$ en $\mathcal{AC}$ zeer frequent voorkomen en $\mathcal{B}$ en $\mathcal{C}$ in UTF-8 dezelfde startbyte hebben. Laat ons coderingen van twee bytes gebruiken, resp. $b_{A_1}b_{A_2}$, $b_S b_{B_2}$ en $b_S b_{C_2}$. In dat geval zal het bytepaar $b_{A_2}b_S$ hoogfrequent zijn, en indien gemerged, is het niet meer mogelijk om de aparte bytegroepen voor $\mathcal{A}$, $\mathcal{B}$ en $\mathcal{C}$ als token te verkrijgen in die strings, wat invloed kan hebben op de keuze van verdere merges. In het beste geval blijft BBPE doormergen tot grotere tokens $b_{A_1}b_{A_2}b_S b_{B_2}$ en $b_{A_1}b_{A_2}b_S b_{C_2}$ die wel karaktergrenzen respecteren.

BPE-dropout OFS zorgt ook dat BPE-dropout minder goede segmentaties suggereert. Om te beginnen volgt het uit het bewijs van OFS dat indien BPE-dropout een merge uitvoert die BPE normaliter niet zou hebben uitgevoerd, de uiteindelijke segmentatie ook niet meer dezelfde kan zijn. Dergelijke merges zijn het enige wat BPE-dropout van BPE onderscheidt, dus is de meerwaarde van BPE-dropout exact het produceren van een andere eindsegmentatie dan BPE.

Welnu, stel dat BPE twee pathologische types samenbrengt en terug interpreteerbaar maakt (bv. *coron+amaatregelen*) in de laatste merge van een woord, dan zorgt BPE-dropout bij eender welke afwijking van BPE voor het opnieuw verschijnen van die afvaltypes. De meerwaarde van BPE-dropout zou juist moeten zijn om *corona*|*maatregelen* aan het model te tonen bij het negeren van de laatste merge.

Overigens kan BPE-dropout het model niet voorbereiden op de segmentaties van de deelwoorden van een samenstelling waarin de lemmagrens overschreden is. Nochtans is het doel van BPE-dropout juist om het model bloot te stellen aan – in de limiet – alle mogelijke segmentaties van een gegeven woord. Beschouw echter de mergeboom die zich in Figuur D.9 uitstrekt boven de letters *maatregelen*: die mergeboom kan nooit ontstaan bij segmentatie van het losse woord *maatregelen*, onafhankelijk van welke merges BPE-dropout wel of niet behoudt. Immers, in geen enkele segmentatie van een deelwoord plakt er een extra karakter voor of achter; dat zou een typo zijn.

4.17.3 *One merge fits all?*

Dat karakters in het midden van een string (“random-access”) kunnen samensmelten, reeds vanaf de eerste BPE-iteratie, is een onoplosbaar gebrek van hiërarchisch mergen.

We zullen moeten opteren voor een andere invoersegmentatie (bv. van links naar rechts), zie §4.7 en §5.2. Echter, bemerk dat de vocabularisatie en invoersegmentatie gelijke bewerkingen uitvoeren op gelijke strings, waardoor ook de vocabularisatiestap van BPE onoplosbaar gebrekkig is.

Een ander perspectief is dat het mergen zelf niet het probleem is, maar waar het gebeurt. In *coronamaatregelen* heeft de *a*+*m*-merge voorrang op de *n*+*a*-merge, en daar gaat het fout. Echter, in *tuinameublement* en *klerkenambt* is dat juist correct, en is het niet erg dat BPE meteen toegang heeft tot het midden van de string. Het probleem is dus eigenlijk dat een BPE-merge op het ganse corpus tegelijk wordt toegepast (zoals een compressiealgoritme), terwijl een merge vaak helemaal niet *one-size-fits-all* is. Vandaar komt Morfessor Cat-MAP (§2.5.2.4) er wél mee weg om hiërarchisch te mergen, want Morfessor segmenteert het corpus *woord per woord* i.p.v. *iteratie per iteratie*.

BPE stelt een contextvrije grammatica op. Een idee kan zijn om een contextgevoelige grammatica op te stellen (met regels die specificeren "merge deze twee tokens alleen als ze omringd zijn door deze andere tokens"), maar het one-size-fits-all-principe zal blijven gelden. Van de oneindig mogelijke samenstellingen is er altijd wel één waarvoor ook contextgevoelige regels voldaan zijn en de merge toch niet mag gebeuren.

Als we wél random-access-merges toelaten en wél een geleerde merge overheen het ganse corpus willen toepassen, dan kan dat mijns inziens alleen maar als de merges niet hard zijn, maar zacht; i.e., dat wanneer een merge $x+y$ wordt uitgevoerd, alle woorden $w\,x\,y\,z$ in het corpus blijven maar met een kleiner gewicht, en dat het overblijvende gewicht naar nieuwe strings $w\,xy\,z$ gaat. Door de informatie van vóór de merge te behouden, is de keuze niet permanent en is er geen informatieverlies. Voor zover ik weet, is er geen literatuur over zachte BPE-merges.

H5 Nieuwe BPE-tokenisers

We zijn nu wegwijs in een resem problemen waar BPE-tokenisers mee te kampen hebben, zoals vooropgesteld in Onderzoeksvraag 1. Meermaals bleek de redenering achter meer gesofisticeerde tokenisers (met name de Morfessor-familie) beter met morfologie overeen te stemmen dan de aannames in BPE; mijns inziens is Hoofdstuk 4 voldoende bewijs om in toekomstige modellen BPE volledig opzij te schuiven en een van de andere tokenisers in §2.5 te kiezen met gunstigere eigenschappen zoals probabilistische segmentaties (§2.5.2.3) en geen "one-size-fits-all"-artefacten (§4.17), of zelfs de tokeniser te verwerpen (§ 2.5.4).

Echter, pre-training van hedendaagse modellen is duur. In §4.9 zagen we dat het partiële of complete hergebruik van de (BPE-)tokeniser van een reeds getraind model toelaat om diens gewichten (bv. de embeddingmatrix) te recycleren. Vandaar dat ik in dit voorlaatste hoofdstuk vertrek vanuit de oorspronkelijke BPE-tokeniser en enkele aanpassingen uitprobeer om problemen uit Hoofdstuk 4 te verhelpen, zoals vooropgesteld in Onderzoeksvraag 2. Ik stel nieuwe tokenisers op volgens twee grote lijnen: andere segmentatiemethodes (§5.2) enerzijds, en processen om kleine aanpassingen aan de mergegraaf te maken anderzijds (§5.3).

5.1 Semi-supervisie

In beide van de volgende secties maak ik gebruik van e-Lex om tokenisers te informeren i.p.v. louter te valideren. Nochtans wordt tokenisatie vandaag steeds (cfr. §2.5) vanuit een ongesuperviseerd perspectief benaderd: tokenisers leren een taal te lezen op basis van doorlopende tekst die geen expliciete hints geeft over de structuur van de woorden. Vandaar is het goed om even te verantwoorden waarom het supplementeren met taalkundige kennis (*semi-supervisie*) acceptabel is.

Ongesuperviseerde tokenisatie is toepasbaar op talen waarvoor geen linguïstische kennis beschikbaar is, of op zijn minst onafhankelijk van het formaat waarin het wel beschikbaar is. Ongesuperviseerde datasets zijn ook in grote, goedkope volumes beschikbaar: in NLP kunnen we genieten van alle publiek beschikbare digitale tekst die ooit geschreven is. Geannoteerde data voor (semi-)supervisie is daarentegen duur, gezien het heel wat werk vraagt om hetzij handmatig – door onderzoekers of door dienstverleners – hetzij door een daartoe gebouwde tool correcte annotaties te bemachtigen. Vandaar ook dat transfer learning recent dominant geworden is.

Echter, men kan zich afvragen of we überhaupt interesse hebben in talen waarvoor

geen morfologische theorie of ontlede dataset bestaat – bv. van een kleine stam in de Sahara, het Amazonewoud of Antarctica. Moest dat een levende taal zijn, dan zijn er noodzakelijkerwijs weinig sprekers van. Moest dat een dode taal zijn, dan zijn er weinig geschreven documenten van. In de tegenovergestelde gevallen zou er anders namelijk reeds academische interesse voor zijn geweest. De enige toepassing van zo'n taal is een vertaalsysteem naar een gangbare taal.

Het grote voordeel van ongesuperviseerd segmenteren is dat het één overkoepelende tokeniser kan gebruiken op een meertalig corpus. Voor kruismodellen zoals ChatGPT-achtige conversationele CLM's is dat een groot voordeel: of de invoercontext nu in het Arabisch of het Nederlands is, de tokenisatie gebeurt hoe dan ook met dezelfde BBPE-tokeniser (§2.5.2.2). Hetzelfde geldt voor de zinnen in het trainingcorpus. Een ander voordeel is dat die ene tokeniser – welk algoritme hij ook gebruikt – kennis van de ene taal kan gebruiken voor de andere, en daarbij aannemelijk een kleiner parameterbudget vergt dan moest elke taal een aparte tokeniser van dezelfde gekregen hebben. Anderzijds zagen we in §4.8 dat zo'n joint-modus voor verwarring kan zorgen en door dataonbalans beïnvloed wordt.

Er is dus motivatie voor zowel semi-supervisie als het gebruik van een vloot van monolinguale tokenisers. Weten wanneer welke tokeniser in de vloot nodig is, is perfect doenbaar: voor monolinguale LM's (bv. RobBERT) is de taal gekend, voor TM's (bv. Google Translate) geeft de gebruiker zelf aan wat de bron- en doeltaal is (en indien nodig wordt gepivoteerd over het Engels), en voor de gevallen zonder gebruikersinterface bestaan er al lang taalidentificeerders met $F_1 > 95\%$, zelfs voor meertalige invoer (bv. Lui e.a., 2014).

Noteer overigens dat Morfessor FlatCat (Grönroos e.a., 2014) en Morfessor EM+Prune (Grönroos e.a., 2020) de optie tot semi-supervisie bieden. Voor BPE-tokenisers komen Huck e.a. (2017) als enige in de buurt van semi-supervisie.

5.2 Andere segmentatiemethodes

5.2.1 Segmentatiemethodes

Algoritme 2.1 toont hoe men het vocabularium en de mergelijst van BPE gebruikt voor segmentatie. We zagen in §4.7 dat het voordeliger kan zijn om vocabularisatie los te koppelen van inferisatie (cfr. §2.5.1.2). Stel dus dat we een vocabularium van subwoordtypes V gegeven krijgen, dan zoeken we andere manieren om het te besteden.

5.2.1.1 Shift-reduce

Een eerste familie van segmenteerders beweegt karakter per karakter van het ene uiteinde van de invoerstring naar het andere. De segmentatiemodule slaat de geconsumeerde karakters op in een buffer (*shift*), en beslist op bepaalde momenten om de kop van de buffer te bundelen tot één token en uit de buffer te wissen (*reduce*).

Stel dat we op zoek zijn naar de grootst mogelijke tokens die we tegenkomen, dan kan dat op een *lazy* wijze of een *greedy* wijze, naar analogie met reguliere expressies. Lazy betekent dat de buffer geleegd wordt bij het eerste karakter dat, indien het aan de buffer wordt toegevoegd, voor een string zorgt die geen type in V is. Dat wil zeggen dat de buffer te allen tijde een subwoord in V is. Greedy blijft consumeren tot het láátste karakter dat voor zo'n string buiten V zou zorgen, waardoor de buffer in de tussentijd even niet in V kan zitten.

Beide zijn van links naar rechts en van rechts naar links toe te passen. We krijgen vier segmenteerders in totaal: L2R-Lazy, R2L-Lazy, L2R-Greedy en R2L-Greedy. Voor een invoerstring van lengte L zijn de lazy varianten $O(L)$ en de greedy varianten $O(L^2)$ in het ergste geval.

Herinner dat Xu e.a. (2021) geen mergelijst terugkregen na vocabularisatie met VOLT. Ze stellen een vijfde shift-reduce-segmenteerder voor:

> After generating the vocabulary, VOLT [...] first splits sentences into character-level tokens. Then, we merge two consecutive tokens into one token if the merged one is in the vocabulary. This process keeps running until no tokens can be merged.

De uitleg is enigszins ambigu, maar aannemelijk werkt hun methode ook van links naar rechts (want er is geen mergelijst, dus geen prioriteit) en bovendien iteratief (want anders was ze gelijk aan L2R-Lazy). Zo implementeer ik ze.

5.2.1.2 Random-access

Aangezien het overschrijden van lemmagrenzen in BPE in de eerste plaats gebeurt doordat er *kleine* merges (die uiteindelijk hiërarchisch uitbreiden tot grotere tokens) worden toegepast op eender welke positie in de string, kan het volstaan om gewoon direct naar grote subwoorden te zoeken. Hoe groter de subwoorden, hoe groter de kans dat ze gehele woorden zijn en lemmagrenzen respecteren.

Daartoe bouw ik nog twee nieuwe segmenteerders. De eerste, RA-Greedy, zoekt eerst doorheen alle $O(L^2)$ substrings naar de langste[67] substring die in V zit. Die substring wordt geïsoleerd als token, en vervolgens past RA-Greedy hetzelfde proces recursief toe op de resterende substrings links en rechts. Bijvoorbeeld: BPE_{nl} splitst het woord *propagandamaatschappijeigenaar* als *Ġpropag|and|am|aat|schapp|ije|igenaar*, maar met hetzelfde vocabularium vindt RA-Greedy eerst het langste token *maatschappij*, en vervolgens ontdekt het dat ook de substrings links en rechts volledige tokens zijn, eindigend bij *Ġpropaganda|maatschappij|eigenaar*.

De tweede segmenteerder, RA-Product, zoekt over alle $O(2^L)$ mogelijke segmentaties diegene met het hoogste product van tokenlengtes. De motivatie is dat dat product incentief geeft om te splitsen, maar niet te veel: zo is $|voetbalmatch| < |voetbal| \cdot |match|$ maar $|voetbal| \cdot |match| > 1^{12}$. De implementatie gebeurt met een viterbialgoritme (zie Appendix B voor gelijkaardige algoritmes) dat het optimum in $O(L^2)$ tijd vindt.

5.2.2 Vocabularisatiemethodes

De bovenstaande segmenteerders hebben alleen nood aan een vooraf bestaand subwoordvocabularium V, geen mergelijst M. Eén manier om V te genereren is met BPE. We kunnen echter ook gebruik maken van e-Lex: in §4.16 kwam het even aan bod dat het niet mogelijk is om BPE-segmentatie morf- of lemmagrenzen te laten respecteren door die grenzen als beperking op te leggen tijdens de leerfase, maar dat lag aan de manier waarop de geleerde V en M vervolgens gebruikt werden. We kunnen diezelfde strategie wel proberen toe te passen als we een van de bovenstaande segmenteerders gebruiken in de plaats.

We hebben dus twee manieren om V te generen: leren zoals Sennrich e.a. (2016) over een onbewerkt corpus als OSCAR-nl, en leren zoals Huck e.a. (2017) met vooraf geplaatste begrenzingen over e-Lex. Een derde manier is nog simpeler: daar we geen nood hebben aan samenhang (i.e. M) tussen de types in V, kunnen we evengoed gewoon alle

[67] Indien er gelijkspel is, wint de meest linkse.

unieke morfen, morfemen of lexen[68] uit e-Lex verzamelen als subwoordvocabularium, en daaraan het alfabet toevoegen ter preventie van ⟨UNK⟩'s.

Ik vergelijk in totaal 12 vocabularia: enerzijds gebruik ik de verzameling van alle morfen, morfemen of lexen (3), de BPE-vocabularia die verkregen worden door elk van die verzamelingen als woorden te behandelen (3) en hetzelfde maar waarbij de lemmata waarvan ze afkomstig zijn eerst gewogen worden met hun OSCAR-frequentie (3). Die laatste zes zijn interessant omdat ze stapstenen van karakters naar de strings in de eerste drie vocabularia genereren en zo beter typo's kunnen opvangen (§4.15).
Anderzijds gebruik ik nog het vocabularium van BPE_{nl} dat afkomstig is van BPE te trainen op OSCAR (1), en de BPE-vocabularia die resulteren door te trainen over de lemmata in e-Lex, gewogen of ongewogen met OSCAR (2). Die laatste twee zijn interessant omdat ze in feite een geïdealiseerde OSCAR modelleren zonder alle rariteiten die normaal voorkomen in een webcorpus.

De 7 tokenisers L2R-Lazy, R2L-Lazy, L2R-Greedy, R2L-Greedy, Xu, RA-Greedy, RA-Product worden voor elk van die 12 vocabularia getest. Bovendien hergebruik ik de 9 BPE-tokenisers (inclusief merges) die bij het genereren van de vocabularia als bijproduct geproduceerd worden. Als laatste voeg ik een karaktertokeniser Alphabet toe om te tonen dat 100% recall mogelijk is (weliswaar met heel lage precision). Dat zijn in totaal 94 tokenisers.

5.2.3 Resultaten

Tabel 5.1 geeft voor alle voorgenoemde tokenisers de precision, recall en F_1 voor morfemische en lexemische splitsingspunten in e-Lex zoals uitgelegd in §4.13. Een aantal vaststellingen:

- Voor de L2R- en R2L-tokenisers is het lazy/greedy dat de trend bepaalt, niet links/rechts. De greedy varianten doet het veel beter.
- L2R-Lazy, R2L-Lazy en Xu presteren alle drie matig omdat ze te vroeg stoppen en te kleine segmenten verkiezen – hetgeen ook aan hun hoge recall te zien is. RA-Product selecteert waarschijnlijk ook te kleine segmenten; zo is het bv. voor $L = 16$ zo dat $16 < 8 \cdot 8 < 4 \cdot 4 \cdot 4 \cdot 4 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$. Er is mogelijks een manier om het product te regulariseren o.b.v. het aantal tokens, opdat er tegendruk tegen het product is.
- De BPE-tokenisers zijn beter dan de bovenstaande, maar kunnen L2R-Greedy, R2L-Greedy en RA-Greedy niet overstijgen.
- Van de BPE-tokenisers is er eentje die het significant slechter doet dan de rest: de middelste (rij O), getraind op OSCAR. Inderdaad, dat is BPE_{nl}. Interessant is wel dat BPE_{nl} competitief is met de L_w-rij, die op een "schone OSCAR" getraind is.
- Hoewel men zou verwachten dat de dataset kopiëren (CC) betere prestaties levert (omdat er minder verwarring is in het kiezen van een token), daalt de performantie van de lazy tokenisers en van Xu bij een vocabularium zonder BPE-tussenstappen.

De winnaars zijn algemeen L2R-Greedy, R2L-Greedy en RA-Greedy met een kopie van de lexen (ℓ) of morfen (m). Van hen drie bevat RA-Greedy de meeste kolommaxima.

[68] Bij gebrek aan betere terminologie noem ik "de samengevoegde morfen die de lexemische splitsing produceert" de "lexen", analoog aan hoe morfemische spliting voor morfen zorgt.

	Dataset	Proces		morfemen						lexemen					
				types			tokens			types			tokens		
			$\|V\|$	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1
Alphabet	–	–	0.1k	0.15	1.00	0.26	0.07	1.00	0.12	0.09	1.00	0.16	0.02	1.00	0.03
L2R-Lazy	L	BPE	39.2k	0.35	0.62	0.45	0.20	0.54	0.29	0.23	0.69	0.34	0.06	0.69	0.12
L2R-Lazy	L_w	BPE	40.1k	0.32	0.66	0.43	0.18	0.57	0.28	0.21	0.73	0.32	0.06	0.72	0.11
L2R-Lazy	M	BPE	16.6k	0.33	0.73	0.46	0.16	0.68	0.26	0.21	0.79	0.33	0.05	0.80	0.09
L2R-Lazy	M_w	BPE	24.5k	0.32	0.71	0.44	0.15	0.65	0.24	0.20	0.77	0.32	0.04	0.77	0.08
L2R-Lazy	O	BPE	40.0k	0.32	0.61	0.42	0.18	0.51	0.27	0.21	0.68	0.32	0.06	0.68	0.11
L2R-Lazy	ℓ	BPE	22.3k	0.34	0.70	0.46	0.17	0.62	0.27	0.23	0.79	0.35	0.06	0.82	0.10
L2R-Lazy	ℓ_w	BPE	40.1k	0.33	0.69	0.45	0.16	0.59	0.25	0.22	0.78	0.34	0.05	0.76	0.10
L2R-Lazy	m	BPE	18.3k	0.37	0.76	0.50	0.18	0.72	0.29	0.22	0.78	0.34	0.05	0.79	0.09
L2R-Lazy	m_w	BPE	30.5k	0.37	0.74	0.50	0.19	0.69	0.30	0.23	0.78	0.35	0.05	0.78	0.10
L2R-Lazy	M	CC	18.9k	0.19	0.88	0.32	0.09	0.87	0.17	0.12	0.90	0.20	0.02	0.92	0.05
L2R-Lazy	ℓ	CC	44.7k	0.17	**0.91**	0.29	0.08	**0.89**	0.15	0.10	0.94	0.19	0.02	**0.95**	0.04
L2R-Lazy	m	CC	25.5k	0.21	0.87	0.34	0.10	0.85	0.19	0.13	0.88	0.22	0.03	0.90	0.05
R2L-Lazy	L	BPE	39.2k	0.35	0.67	0.46	0.22	0.65	0.33	0.23	0.75	0.35	0.07	0.81	0.12
R2L-Lazy	L_w	BPE	40.1k	0.33	0.71	0.45	0.20	0.66	0.31	0.21	0.79	0.33	0.06	0.83	0.12
R2L-Lazy	M	BPE	16.6k	0.37	0.78	0.50	0.21	0.74	0.33	0.24	0.85	0.37	0.06	0.88	0.12
R2L-Lazy	M_w	BPE	24.5k	0.35	0.75	0.47	0.19	0.69	0.30	0.23	0.83	0.35	0.06	0.86	0.11
R2L-Lazy	O	BPE	40.0k	0.35	0.64	0.46	0.26	0.58	0.36	0.23	0.72	0.35	0.09	0.78	0.16
R2L-Lazy	ℓ	BPE	22.3k	0.36	0.72	0.48	0.21	0.66	0.32	0.24	0.84	0.38	0.07	0.88	0.13
R2L-Lazy	ℓ_w	BPE	40.1k	0.34	0.73	0.46	0.21	0.69	0.32	0.23	0.83	0.36	0.07	0.87	0.12
R2L-Lazy	m	BPE	18.3k	0.37	0.75	0.50	0.22	0.69	0.33	0.24	0.84	0.38	0.07	0.88	0.13
R2L-Lazy	m_w	BPE	30.5k	0.37	0.76	0.50	0.23	0.73	0.35	0.23	0.83	0.37	0.07	0.86	0.13
R2L-Lazy	M	CC	18.9k	0.20	0.87	0.32	0.10	0.83	0.17	0.12	0.92	0.21	0.03	0.93	0.05
R2L-Lazy	ℓ	CC	44.7k	0.18	**0.91**	0.30	0.09	**0.89**	0.16	0.11	0.93	0.19	0.02	0.94	0.04
R2L-Lazy	m	CC	25.5k	0.22	0.83	0.35	0.12	0.80	0.21	0.14	0.89	0.24	0.03	0.91	0.07
RA-Product	L	BPE	39.2k	0.31	0.63	0.41	0.16	0.60	0.25	0.19	0.67	0.30	0.04	0.67	0.08
RA-Product	L_w	BPE	40.1k	0.32	0.66	0.43	0.16	0.63	0.26	0.20	0.70	0.31	0.05	0.72	0.09
RA-Product	M	BPE	16.6k	0.34	0.71	0.46	0.17	0.68	0.27	0.21	0.76	0.33	0.05	0.76	0.09
RA-Product	M_w	BPE	24.5k	0.33	0.70	0.45	0.16	0.66	0.26	0.21	0.74	0.32	0.05	0.74	0.09
RA-Product	O	BPE	40.0k	0.30	0.62	0.40	0.15	0.59	0.24	0.18	0.64	0.28	0.04	0.64	0.08
RA-Product	ℓ	BPE	22.3k	0.33	0.68	0.44	0.16	0.64	0.26	0.21	0.74	0.33	0.05	0.75	0.09
RA-Product	ℓ_w	BPE	40.1k	0.33	0.68	0.44	0.16	0.63	0.26	0.21	0.74	0.32	0.05	0.74	0.09
RA-Product	m	BPE	18.3k	0.34	0.71	0.46	0.17	0.69	0.28	0.21	0.73	0.32	0.05	0.74	0.09
RA-Product	m_w	BPE	30.5k	0.34	0.70	0.46	0.17	0.69	0.28	0.20	0.72	0.32	0.05	0.74	0.09
RA-Product	M	CC	18.9k	0.47	0.89	0.62	0.24	0.85	0.37	0.29	0.92	0.44	0.07	**0.95**	0.12
RA-Product	ℓ	CC	44.7k	0.51	0.83	0.63	0.26	0.76	0.39	0.33	0.93	0.49	0.08	**0.95**	0.15
RA-Product	m	CC	25.5k	0.49	0.88	0.63	0.26	0.86	0.40	0.29	0.90	0.44	0.07	0.93	0.13
Xu	L	BPE	39.2k	0.49	0.45	0.47	0.25	0.26	0.25	0.38	0.61	0.47	0.11	0.47	0.18
Xu	L_w	BPE	40.1k	0.42	0.49	0.45	0.16	0.09	0.12	0.31	0.63	0.42	0.08	0.20	0.12
Xu	M	BPE	16.6k	0.49	0.72	0.58	0.29	0.64	0.40	0.31	0.78	0.44	0.09	0.78	0.16
Xu	M_w	BPE	24.5k	0.48	0.69	0.56	0.29	0.59	0.39	0.31	0.76	0.44	0.09	0.74	0.16
Xu	O	BPE	40.0k	0.41	0.51	0.45	0.25	0.13	0.17	0.30	0.65	0.41	0.15	0.32	0.21
Xu	ℓ	BPE	22.3k	0.49	0.56	0.52	0.24	0.35	0.29	0.40	0.78	0.53	0.13	0.78	0.23
Xu	ℓ_w	BPE	40.1k	0.47	0.51	0.49	0.27	0.21	0.24	0.41	0.75	0.53	0.23	0.71	0.35
Xu	m	BPE	18.3k	0.56	0.74	0.64	0.35	0.70	0.47	0.34	0.77	0.47	0.10	0.77	0.17
Xu	m_w	BPE	30.5k	0.57	0.72	0.64	0.49	0.65	0.56	0.35	0.76	0.48	0.14	0.74	0.24
Xu	M	CC	18.9k	0.21	0.82	0.34	0.11	0.79	0.19	0.13	0.88	0.23	0.03	0.90	0.06
Xu	ℓ	CC	44.7k	0.19	0.87	0.31	0.09	0.83	0.17	0.11	0.91	0.20	0.03	0.92	0.05
Xu	m	CC	25.5k	0.26	0.80	0.39	0.16	0.77	0.26	0.16	0.85	0.27	0.04	0.86	0.08
BPE	L	BPE	39.2k	0.69	0.48	0.56	0.39	0.30	0.34	0.54	0.65	0.59	0.17	0.52	0.25
BPE	L_w	BPE	40.1k	0.56	0.49	0.52	0.60	0.03	0.07	0.44	0.66	0.53	0.47	0.11	0.17
BPE	M	BPE	16.6k	0.64	0.81	0.72	0.44	0.80	0.57	0.39	0.86	0.54	0.12	0.86	0.21
BPE	M_w	BPE	24.5k	0.65	0.79	0.72	0.47	0.78	0.59	0.40	0.84	0.55	0.13	0.85	0.22
BPE	O	BPE	40.0k	0.52	0.55	0.54	0.55	0.12	0.19	0.39	0.70	0.50	0.36	0.31	0.33
BPE	ℓ	BPE	22.3k	0.66	0.58	0.62	0.33	0.38	0.36	0.56	0.84	0.67	0.18	0.83	0.30
BPE	ℓ_w	BPE	40.1k	0.66	0.52	0.58	0.82	0.21	0.33	0.60	0.81	0.69	0.82	0.82	0.82
BPE	m	BPE	18.3k	0.71	0.82	0.76	0.49	0.81	0.61	0.43	0.85	0.57	0.13	0.84	0.22
BPE	m_w	BPE	30.5k	0.76	0.80	0.78	0.86	0.84	0.85	0.45	0.82	0.58	0.22	0.85	0.35
L2R-Greedy	L	BPE	39.2k	0.68	0.47	0.56	0.32	0.26	0.28	0.55	0.66	0.60	0.15	0.47	0.22
L2R-Greedy	L_w	BPE	40.1k	0.58	0.49	0.53	0.62	0.03	0.07	0.47	0.68	0.56	0.48	0.11	0.18
L2R-Greedy	M	BPE	16.6k	0.67	0.80	0.73	0.41	0.73	0.53	0.42	0.87	0.57	0.13	0.89	0.22
L2R-Greedy	M_w	BPE	24.5k	0.68	0.77	0.72	0.42	0.67	0.52	0.44	0.86	0.58	0.13	0.85	0.23
L2R-Greedy	O	BPE	40.0k	0.53	0.54	0.54	0.54	0.11	0.19	0.41	0.72	0.52	0.38	0.32	0.34
L2R-Greedy	ℓ	BPE	22.3k	0.69	0.59	0.64	0.31	0.36	0.33	0.59	0.87	0.70	0.19	0.85	0.31
L2R-Greedy	ℓ_w	BPE	40.1k	0.73	0.54	0.63	0.86	0.20	0.33	0.67	0.86	0.75	0.85	0.81	0.83
L2R-Greedy	m	BPE	18.3k	0.77	0.84	0.80	0.49	0.80	0.60	0.46	0.88	0.61	0.13	0.88	0.23
L2R-Greedy	m_w	BPE	30.5k	0.82	0.82	0.82	0.77	0.75	0.76	0.50	0.86	0.64	0.22	0.84	0.35
L2R-Greedy	M	CC	18.9k	0.66	0.87	0.75	0.39	0.78	0.52	0.41	0.93	0.57	0.11	0.90	0.20
L2R-Greedy	ℓ	CC	44.7k	0.93	0.55	0.69	0.87	0.23	0.36	0.93	0.93	0.93	0.87	0.92	0.89
L2R-Greedy	m	CC	25.5k	0.89	0.90	0.89	0.83	0.84	0.83	0.53	0.92	0.68	0.22	0.89	0.35
R2L-Greedy	L	BPE	39.2k	0.68	0.48	0.56	0.50	0.26	0.34	0.55	0.66	0.60	0.25	0.52	0.34
R2L-Greedy	L_w	BPE	40.1k	0.57	0.48	0.52	0.61	0.03	0.07	0.46	0.67	0.54	0.48	0.11	0.18
R2L-Greedy	M	BPE	16.6k	0.58	0.73	0.65	0.36	0.63	0.46	0.39	0.85	0.54	0.12	0.84	0.21
R2L-Greedy	M_w	BPE	24.5k	0.60	0.72	0.65	0.37	0.60	0.46	0.41	0.84	0.55	0.13	0.83	0.22
R2L-Greedy	O	BPE	40.0k	0.53	0.54	0.53	0.56	0.12	0.19	0.41	0.71	0.52	0.39	0.32	0.35
R2L-Greedy	ℓ	BPE	22.3k	0.71	0.59	0.64	0.50	0.35	0.41	0.60	0.85	0.70	0.30	0.87	0.45
R2L-Greedy	ℓ_w	BPE	40.1k	0.73	0.54	0.62	0.93	0.22	0.35	0.67	0.85	0.75	0.92	0.86	0.89
R2L-Greedy	m	BPE	18.3k	0.68	0.76	0.72	0.55	0.69	0.61	0.45	0.85	0.59	0.17	0.86	0.28
R2L-Greedy	m_w	BPE	30.5k	0.75	0.75	0.75	0.71	0.69	0.70	0.49	0.84	0.62	0.22	0.85	0.35
R2L-Greedy	M	CC	18.9k	0.61	0.85	0.71	0.37	0.78	0.50	0.39	0.93	0.55	0.11	0.91	0.19
R2L-Greedy	ℓ	CC	44.7k	0.93	0.54	0.68	**0.96**	0.23	0.37	0.92	0.92	0.92	**0.96**	0.92	**0.94**
R2L-Greedy	m	CC	25.5k	0.78	0.83	0.80	0.73	0.77	0.75	0.49	0.90	0.64	0.21	0.90	0.34
RA-Greedy	L	BPE	39.2k	0.70	0.47	0.57	0.42	0.22	0.29	0.56	0.65	0.61	0.22	0.47	0.30
RA-Greedy	L_w	BPE	40.1k	0.61	0.50	0.55	0.64	0.04	0.07	0.49	0.70	0.58	0.50	0.11	0.18
RA-Greedy	M	BPE	16.6k	0.71	0.83	0.77	0.45	0.72	0.55	0.46	0.94	0.62	0.14	0.91	0.24
RA-Greedy	M_w	BPE	24.5k	0.73	0.82	0.77	0.45	0.68	0.54	0.49	0.93	0.64	0.15	0.89	0.25
RA-Greedy	O	BPE	40.0k	0.56	0.57	0.56	0.58	0.12	0.19	0.44	0.76	0.56	0.40	0.32	0.36
RA-Greedy	ℓ	BPE	22.3k	0.79	0.63	0.70	0.47	0.34	0.40	0.68	0.93	0.79	0.30	0.88	0.45
RA-Greedy	ℓ_w	BPE	40.1k	0.80	0.57	0.67	0.92	0.21	0.34	0.74	0.91	0.82	0.92	0.84	0.88
RA-Greedy	m	BPE	18.3k	0.84	0.88	0.86	0.65	0.80	0.72	0.52	0.94	0.67	0.18	0.91	0.30
RA-Greedy	m_w	BPE	30.5k	0.90	0.86	0.88	0.82	0.76	0.79	0.56	0.92	0.70	0.24	0.88	0.37
RA-Greedy	M	CC	18.9k	0.69	0.89	0.78	0.42	0.79	0.55	0.43	**0.95**	0.60	0.12	0.91	0.21
RA-Greedy	ℓ	CC	44.7k	**0.97**	0.55	0.70	0.92	0.23	0.37	**0.97**	0.94	**0.95**	0.92	0.91	0.91
RA-Greedy	m	CC	25.5k	0.93	0.90	**0.92**	0.88	0.85	**0.86**	0.56	0.94	0.70	0.23	0.90	0.37

Tabel 5.1 – Evaluatie van nieuwe tokenisers op splitsingspunten in e-Lex. L = lemmata, M = morfemen, O = OSCAR, ℓ = lexen, m = morfen, w = gewogen, CC = kopieer.

5.3 BPE-knockout

In de vorige sectie hebben we de mergelijst M die de BPE-leerfase produceert, volledig genegeerd. Hoewel die een pijnpunt is van BPE, zit er uiteraard alsnog nuttige informatie in vervat. Vandaar dat ik in deze sectie een nieuwe methode voorstel, *BPE-knockout*, die de mergelijst aanpast op zo'n manier dat ze grotendeels hetzelfde blijft maar BPE-segmentatie toch morfologischer wordt. In de volgende subsecties bouw ik dat systeem component per component op.

5.3.1 Mergegraaf

De mergelijst M heeft op zich een platte structuur, maar zoals Figuur 1.2 toont, is het mogelijk om er een gerichte graaf (*digraaf*) van te maken. Immers, een mergeregel $x + y \rightarrow xy$ refereert impliciet aan de twee mergeregels die x resp. y produceerden, en is ook de aanleiding tot alle mergeregels die het nieuwe type xy gebruiken. Die twee relaties zijn respectievelijk de inkomende en uitgaande bogen in de mergegraaf.

Noteer dat M die digraaf definieert, maar niet door alleen de verzameling van vertices en verzameling bogen vervangen kan worden. Een graaf heeft nl. niet zoiets als een volgorde waarin de bogen geplaatst worden, noch een volgorde waarin bogen aankomen. Om een merge $x + y \rightarrow z$ voor te stellen is het niet voldoende dat er vertices $\{x, y, z\}$ en bogen $\{(x, z), (y, z)\}$ zijn; volledieger is de voorstelling $(i, (x, y))$ met i de index die (x, y) had in M. Het is niet nodig om z expliciet in de merge te vermelden daar $z = xy$.

Wat wel kan, is M voorstellen met een digraaf waarvan de vertices afwisselend een type en een merge zijn. Elke merge wijst naar het type dat het produceert, en elk type wijst naar de merges waaraan het deelneemt. A.d.h.v. een verzameling merge-objecten $\mathcal{M} = \{(i, (x, y))\}$ is die afwisselende graaf ook voor te stellen met twee lijsten $M_i(\mathfrak{t})$ en $M_o(\mathfrak{t})$ per type $\mathfrak{t}$, met daarin alle referenties naar zijn inkomende resp. uitgaande merges.

5.3.2 BTE

In klassieke BPE gelden twee invarianten: de ene is dat elke merge exact twee ouders betrekt ($\forall (i, m) \in \mathcal{M} : |m| = 2$) en de andere is dat elk type door exact één merge gevormd wordt ($\forall \mathfrak{t} \in V : |M_i(t)| = 1$), uit §4.17 bekend als (een gevolg van) OFS.

De eerste invariant spiegelt het algoritme van Gage (1994) en is nuttig om de complexiteit van lijn 7 van Algoritme 2.1 te drukken, maar is niet noodzakelijk. Een merge $(i, (a, b, c))$ ter vorming van het type *abc* is, eens hij in de graaf bestaat, perfect compatibel met bestaande BPE-segmentatie. In dat geval kunnen we in plaats van "byte-pair encoding" spreken van *byte-tuple encoding (BTE)*.

De tweede invariant werd in §4.17 verklaard door het verdwijnen van alle data die een alternatieve merge hadden kunnen suggereren voor een reeds gevormd type; om dat op te lossen stelde ik daar reeds voor om ofwel de segmentatiemethode aan te passen (wat we hierboven reeds deden), ofwel om met zachte merges te werken.
Een derde oplossing om de impact van OFS te verminderen – maar niet te elimineren – is om M intelligent te herordenen. Immers, als de frequentie van een tokenpaar (x, y) enorm hoog is, wijst dat er misschien juist op dat dat tokenpaar in *abnormaal veel* woorden voorkomt en dus juist mínder mergeprioriteit moet krijgen vanwege de grote kans op grensoverschrijdend gedrag. In plaats daarvan laten we beter andere merges voorgaan die token x naar links trekken of token y naar rechts, waardoor de merge (x, y) minder schade aanricht. De implementatie van dat idee valt buiten de scope van deze thesis.

5.3.3 Knockout

In §4.10 en §4.11 zagen we dat BPE veel nutteloze types creëert als stapsteen naar grotere, bruikbare types. Nu we meer dan binaire merges kunnen uitvoeren dankzij BTE, zijn die stapstenen overbodig.

Eén manier om daarvan gebruik te maken, is om lijn 7 van Algoritme 2.1 niet alleen tokenparen te laten tellen, maar ook tupels van hogere orde. Op die manier omzeilen we stapstenen, omdat we meteen grote tokens kunnen vormen. Echter, dat zorgt voor een combinatorische explosie, waardoor het zeer veel tijd en geheugen vergt; daarbij hebben dergelijke tupels (N-grammen) ook een spaarsheidsprobleem, waardoor aantallen onbetrouwbaar worden.

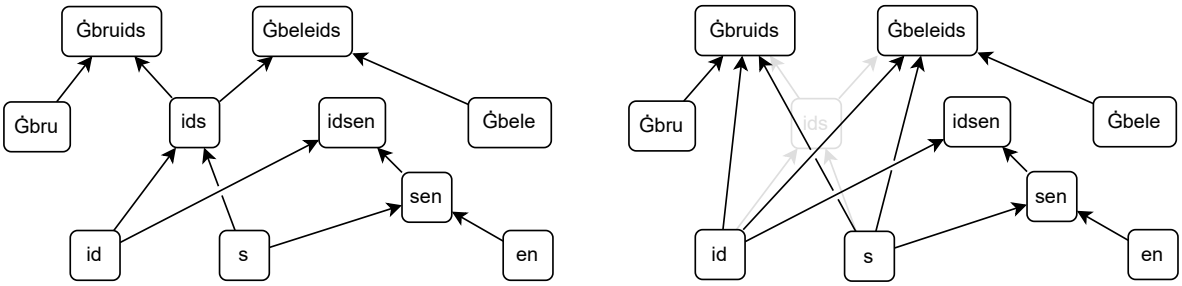
Het alternatief is om stapsteentypes te verwijderen uit een bestaande, binaire BPE-tokeniser. Het voordeel van types louter te verwijderen is dat modellen getraind met die tokeniser geen bijkomende embeddings nodig hebben, en er zelfs voordeel uit halen van een kleinere embedding-LUT te hebben (cfr. § 4.12.4).

Zij dus gegeven de binaire mergegraaf (V, M_i, M_o) geproduceerd door klassieke BPE. Door een type $\mathfrak{t}$ uit de graaf te verwijderen, worden de merge(s) in $M_i(\mathfrak{t})$ illegaal (want het resultaat zit niet meer in V) en wordt het onmogelijk om de merges in $M_o(\mathfrak{t})$ uit te voeren (want een van hun ouders is niet meer vormbaar). Door OFS hebben alle resulterende types van $M_o(\mathfrak{t})$ slechts één merge die hen vormt, dus kunnen ze *tout court* niet meer gevormd worden, wat recursief doorheen de mergegraaf cascadeert. Dat is duidelijk ongewenst. Een beter idee is om het type te overbruggen door zijn uitgaande bogen opnieuw toe te wijzen, maar aan zijn ouders. De volledige *knockout*-procedure is geformaliseerd in Algoritme 5.1.

Algoritme 5.1 Knockout: verwijdering van een type uit de BPE-mergegraaf

1: **function** KNOCKOUT(V, M_i, M_o, $\mathfrak{t}$)
2: **if** $|M_i(\mathfrak{t})| = 0$ **then** ▷ $\mathfrak{t}$ deel van het alfabet
3: **return** (V, M_i, M_o)
4: $\{m_{\text{old}}\} \leftarrow M_i(\mathfrak{t})$
5: $o_1, \ldots, o_n \leftarrow$ OUDERS(m_{old})
6: **for** $i \in 1 \ldots n$ **do**
7: $M_o(o_i) \leftarrow M_o(o_i) \setminus \{m_{\text{old}}\}$ ▷ elke ouder vormt $\mathfrak{t}$ niet meer
8: **for all** $(q, (t_1, \ldots, \mathfrak{t}, \ldots, t_m)) \in M_o(\mathfrak{t})$ **do**
9: $m_{\text{new}} \leftarrow (q, (t_1, \ldots, o_1, \ldots, o_n, \ldots, t_m))$
10: $M_i(t_1 \ldots \mathfrak{t} \ldots t_m) \leftarrow \{m_{\text{new}}\}$ ▷ kind van $\mathfrak{t}$ gevormd door nieuwe merge
11: **for** $i \in 1 \ldots n$ **do**
12: $M_o(o_i) \leftarrow M_o(o_i) \cup \{m_{\text{new}}\}$ ▷ elke ouder van $\mathfrak{t}$ vormt nu dit kind
13: $V \leftarrow V \setminus \{\mathfrak{t}\}$
14: $M_i(\mathfrak{t}) \leftarrow \{\}$
15: $M_o(\mathfrak{t}) \leftarrow \{\}$
16: **return** (V, M_i, M_o)

Figuur 5.1 toont een deel van de BPE_{nl}-mergraaf rond het type *ids*, en wat er gebeurt als dat type *ids* uit de graaf geslagen wordt. Merk bijvoorbeeld op dat BTE het type *Ġbruids* vormt met een triomerge *Ġbru+id+s*, waarin de *s* een interfix is.



Figuur 5.1 – Visualisatie van de veranderingen in een deel van de BPE_{nl}-mergegraaf na knockout van het type *ids*, afkomstig van de merge *id+s*.

5.3.4 Blame

Het is natuurlijk suboptimaal om willekeurige types uit de mergegraaf te slaan, dus hebben we een metriek nodig om te weten voor welke types dat voordelig is. Omdat OFS een invariant is voor én na knockout – elk type wordt door exact één merge gevormd – is het equivalent om ons af te vragen welke merge (i.p.v. welk type) beter verdwijnt.

Herinner dat we tijdens morfologische experimenten met e-Lex vergeleken welke splitsingspunten de tokeniser had moeten plaatsen (referenties/positieven), welke hij had geplaatst (predicties), en welke daarvan overeenkwamen (echte positieven). Er zijn twee soorten fouten te maken (zie §4.13): valse positieven, nl. splitsingen die de tokeniser plaatste terwijl er een samentrekking moest gebeuren, en valse negatieven, nl. samentrekkingen die de tokeniser maakte terwijl er eigenlijk een splitsing gelaten moest zijn.

Voordat we aan knockout doen, zijn er alleen binaire merges aanwezig, die elk exact één splitsing tussen twee karakters samentrekken: dat betekent dat we de schuld (Eng.: *blame*) van elke valse negatieve op één merge kunnen steken. Een merge die veel beschuldigd wordt van incorrect grenzen te overschrijden is een ongewenste merge, en geschikt voor knockout. Welnu: we voeren met de oorspronkelijke BPE-tokeniser een morfologische vergelijking uit op e-Lex zoals gewoonlijk, maar houden voor elke merge m bij hoe vaak hij gebruikt is ($N(m)$) en hoe vaak hij beschuldigd wordt ($B(m)$) van valse negatieven te produceren. De verhouding

$$R(m) = \frac{B(m)}{N(m)} \tag{5.1}$$

noem ik de schuldverhouding (Eng.: *blame ratio*). Merges met $R(m) > \frac{1}{2}$ zijn meer slecht dan goed, en worden zodus geselecteerd voor knockout. De volgorde waarin dergelijke types verwijderd worden, maakt niet uit.[69]

Zoals gewoonlijk kunnen we een morfemische splitsing of lexemische splitsing doen. Welke we gebruiken, is een hyperparameter van BPE-knockout, die ik de *modus* noem.[70]

Voor-en-na Laat ons om te beginnen BPE-knockout toepassen op BPE_{nl} en hun splitsingspunten vergelijken.

- In lexemische modus blijft 96% van de oude splitsingen behouden (recall), en 90% van de gemaakte splitsingen werd reeds gemaakt (precision). Het effect is duidelijk: BPE-knockout voegt extra splitsingen toe. In 10% van e-Lex-lemmata verschijnen er meer tokens dan voordien.

[69] Het bewijs wordt aan de lezer gelaten; het komt opnieuw neer op de invariantie van OFS.

[70] Bij gebrek aan tijd heb ik alleen met *ongewogen* splitsingen kunnen werken. Anders waren er twee extra modi geweest: lexemische en morfemische splitsingen gewogen met het gewicht van hun lemma in OSCAR, zoals gedaan wordt in de "token"-kolom van Tabel 5.1.

- In morfemische modus blijft 93% van oude splitsingen behouden, maar is slechts 77% al gemaakt. Nu hebben 25% van e-Lex-lemmata meer tokens dan voordoen.

Figuur D.19 toont de volledige verdelingen van tokenverschillen. Het verschil tussen de twee is kennelijk dat de morfemische modus veel agressiever splitsingen toevoegt, d.w.z. merges uit de graaf slaat. Dat houdt steek, want de lexemische modus kan maar een deelverzameling van alle morfemische splitsingspunten zien en blame toewijzen.

Geselecteerde merges Wat voor merges zijn geselecteerd voor knockout? Figuur D.20 toont de verdelingen van het aantal karakters van het type links t_ℓ en het type rechts t_r in de verwijderde merges $t_\ell + t_r$. Opmerkelijk is dat de gemiddeldes van de verdelingen overeenkomen voor de lexemische en morfemische modus: beide kiezen merges met gemiddeld 5 karakters links en 4 karakters rechts. De piek bij rechtertypelengte 1 voor de lexemische modus slaat logischerwijs op interfices, waarbij vervolgens lengtes 2 en 3 minder voorkomen omdat lemmata veelal groter zijn dan dat. Wat ook opmerkelijk is aan de vier verdelingen, is dat ze niet overeenkomen met de verdeling van alle merges in BPE_{nl} (Figuur D.21). Dat wil zeggen dat BPE-knockout wel degelijk iets leert en niet gewoon een willekeurig sample aan merges neemt.

Figuur D.22 toont de prioriteiten van de verwijderde merges. De verdeling is in beide modi redelijk uniform.

Triviale merges We willen de twee modi graag evalueren zoals we in §5.2 deden. Dat willen we echter zo eerlijk mogelijk doen. Een van de dingen die BPE-knockout doet, is samenstellingen die als type voorkomen in BPE_{nl} terug uiteenhalen. De merge die een samenstelling produceert, bestaat alleen maar om die ene samenstelling te vormen, i.t.t. generieke merges zoals van twee karakters. e-Lex eist echter dat er een splitsing staat tussen de delen van een samenstelling (wat correct is), dus is zo'n merge altijd fout (1 van de 1 keren dat hij wordt toegepast) en is $R(m) = 1$. BPE-knockout leert in dergelijke gevallen e-Lex van buiten, waardoor het gratis recall krijgt.

De bedoeling van BPE-knockout is om patronen overheen veel voorbeelden te leren, niet om e-Lex te memoriseren. Het is dus goed om te weten hoe BPE-knockout presteert zónder zulke triviale merges te verwijderen – het zijn alleen de resterende merges waar de echte veralgemenende capaciteit van BPE-knockout zit. In de praktijk is het moeilijk om triviale merges exact te detecteren; $N(m) = 1$ vangt lang niet alle triviale merges op, aangezien een samenstelling soms in een genestelde samenstelling voorkomt. Ik gebruik de heuristiek dat een merge triviaal is wanneer beide types minstens lengte 4 hebben. De lexemische modus vond oorspronkelijk 1597 kandidaten om te verwijderen ($R(m) > \frac{1}{2}$), maar 1048 daarvan zijn triviaal volgens die heuristiek. De morfemische modus vond er 4299 waarvan 2041 triviaal.

Resultaten De resultaten van die triviale merges wel of niet te verwijderen zijn in Tabel 5.2 te zien. Er is een merkbaar performantieverlies door geen gebruik te maken van de triviale merges, zoals voorspeld. Echter, BPE-knockout doet het nog steeds beter dan BPE in alle kolommen zelfs zonder triviale merges (met één uitzondering, nl. de precision van morfemische modus op gewogen lexemische splitsingspunten, vanwege zijn agressieve splitsingsgedrag zoals hierboven besproken).

5.3.5 Annealing

BPE-knockout bevrijdt sommige tokens uit een slechte merge, wat toelaat om merges in de andere richting te leren met die vrijgekomen tokens – hetgeen doet denken aan

	Knockout	Anneal	$\lvert V\rvert$	morfemen						lexemen					
				types			tokens			types			tokens		
				Pr	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1
BTE	–	–	40.0k	0.52	0.55	0.54	0.55	0.12	0.19	0.38	0.70	0.50	0.36	0.31	0.33
BTE-knockout	L	–	38.4k	0.56	0.62	0.59	0.68	0.24	0.35	**0.43**	0.82	**0.56**	**0.56**	0.79	**0.66**
BTE-knockout-notrivial	L	–	39.5k	0.55	0.61	0.58	0.57	0.15	0.23	0.42	0.80	0.55	0.41	0.42	0.41
BTE-knockout	M	–	35.7k	**0.61**	**0.78**	**0.69**	**0.82**	**0.65**	**0.72**	0.38	**0.83**	0.52	0.26	**0.82**	0.39
BTE-knockout-notrivial	M	–	37.7k	0.60	0.73	0.66	0.72	0.38	0.49	0.38	0.80	0.51	0.20	0.42	0.28

Tabel 5.2 – Evaluatie van BPE-knockout op morfemische en lexemische splitsingspunten in e-Lex.

een *annealing*-proces in de materiaalkunde waarbij een metaalrooster zichzelf langzaam van dislocaties ontdoet. Om dat proces in goede banen te leiden, beschouwen we niet álle mogelijke tokenparen, want dat zou neerkomen op BPE; in plaats daarvan maken we gebruik van de valse positieven in e-Lex. Die splitsingen waarvan e-Lex beweert dat ze niet nodig zijn, zijn de enige kandidaten voor nieuwe merges.

Noteer dat het moeilijker is om schuld toe te kennen in de context van valse positieven, aangezien een gat tussen twee karakters niet de schuld is van één merge die wel gebeurd is, maar juist een hele verzameling merges die nooit gebeurd zijn tussen de steeds groeiende tokens aan weerszijden van het gat. Door alleen valse positieven te meten *nadat* de segmentatie van een lemma volledig is uitgevoerd, missen we de informatie over merges die in het midden van de historiek best waren uitgevoerd.

Bovendien is het zoeken naar typeparen om samen te voegen (annealing) een veel spaarser proces dan zoeken naar bestaande types om te verwijderen (knockout), vanwege de kwadratische zoekruimte. Vandaar hou ik alleen het absolute aantal keer $A(m)$ bij dat een merge valse positieven oplost, en voer ik de merge uit als $A(m) > 25$ (handmatig gekozen o.b.v. de observatie dat lagere $A(m)$ over enkelvoudige woordstammen gaat).

Ook annealing heeft twee modi. We kunnen bovendien kiezen of we eerst annealing uitvoeren, of eerst knockout.

Resultaten Tabel 5.3 toont opnieuw een morfologische evaluatie, dit keer voor BPE-knockout met of zonder annealing, en annealing met of zonder knockout, voor elke combinatie van modi. A.d.h.v. de blokstructuur zien we dat de performantie weinig van annealing afhangt en vooral van de knockoutmodus.[71]

[71] Aan de vocabulariumgrootte van de eerste drie rijen te zien, ligt het probleem waarschijnlijk bij een te hoge drempel voor $A(m)$, waardoor er bijna geen annealing gebeurt. Bij gebrek aan tijd kon ik die hyperparameter helaas niet tunen.

	Knockout	Annealing	$\lvert V\rvert$	morfemen						lexemen					
				types			tokens			types			tokens		
				Pr	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1	**Pr**	**Re**	F_1
BTE	–	–	40.0k	0.52	0.55	0.54	0.55	0.12	0.19	0.38	0.70	0.50	0.36	0.31	0.33
BTE-anneal	–	M	40.0k	0.53	0.55	0.54	0.55	0.12	0.19	0.39	0.70	0.50	0.37	0.31	0.33
BTE-anneal	–	L	40.0k	0.53	0.55	0.54	0.55	0.12	0.19	0.39	0.70	0.50	0.37	0.31	0.33
BTE-knockout	L	–	38.4k	0.56	0.62	0.59	0.68	0.24	0.35	0.43	0.82	0.56	0.56	0.79	**0.66**
BTE-knockout-anneal	L	M	38.5k	0.56	0.62	0.59	0.69	0.24	0.36	0.43	0.82	**0.57**	**0.57**	0.79	**0.66**
BTE-anneal-knockout	L	M	38.4k	0.56	0.62	0.59	0.69	0.24	0.35	0.43	0.82	**0.57**	0.56	0.79	**0.66**
BTE-knockout-anneal	L	L	38.5k	0.56	0.62	0.59	0.69	0.24	0.36	**0.44**	0.82	**0.57**	**0.57**	0.79	**0.66**
BTE-anneal-knockout	L	L	38.5k	0.56	0.62	0.59	0.69	0.24	0.35	0.43	0.82	**0.57**	0.56	0.79	**0.66**
BTE-knockout	M	–	35.7k	0.61	**0.78**	**0.69**	**0.82**	**0.65**	**0.72**	0.38	**0.83**	0.52	0.26	**0.82**	0.39
BTE-knockout-anneal	M	M	35.8k	**0.62**	**0.78**	**0.69**	**0.82**	0.64	**0.72**	0.39	**0.83**	0.53	0.26	**0.82**	0.40
BTE-anneal-knockout	M	M	35.7k	**0.62**	**0.78**	**0.69**	**0.82**	**0.65**	**0.72**	0.38	**0.83**	0.52	0.26	**0.82**	0.39
BTE-knockout-anneal	M	L	35.9k	0.61	0.72	0.66	0.81	0.59	0.68	0.41	**0.83**	0.54	0.28	**0.82**	0.42
BTE-anneal-knockout	M	L	35.7k	**0.62**	**0.78**	**0.69**	**0.82**	**0.65**	**0.72**	0.38	**0.83**	0.52	0.26	**0.82**	0.39

Tabel 5.3 – Evaluatie van BPE-knockout en BPE-annealing op morfemische (links) en lexemische (rechts) splitsingspunten in e-Lex, gewogen met OSCAR (tokens) of niet (types). "L" wijst op de lexemische modus, "M" op de morfemische.

H6 Conclusie

Om de thesis af te sluiten, som ik in dit laatste hoofdstuk de belangrijkste ideeën uit de voorgaande hoofdstukken op. Vermits er in mijn betoog op allerlei vlakken twijfels over BPE-tokenisers naar boven kwam drijven, eindig ik met een lijst van *future work*.

6.1 Samengevat

Zoals in het abstract staat, had deze thesis drie doelen: een referentiekader scheppen over subwoordtokenisatie (Hoofdstuk 1 en Hoofdstuk 2), een compilatie en analyse maken van de kritieken over BPE (Hoofdstuk 4), en enkele remedies voorstellen (Hoofdstuk 5).

Hoofdstuk 1 besprak tekstverwerking *ab ovo*. Enkele sleutelconcepten uit taalkunde (lexemen, interfices, lexicalisering ...) en uit NLP (vocabularium, tokeniser, token vs. type ...) werden uiteengezet, waaronder het terugkerende idee van blinde interfaces: taalmodellen weten niet wat tekst is. Subwoordtokenisatie werd gemotiveerd als remedie tegen enerzijds het open woordvocabularium van de taal, en anderzijds de spaarsheid van taaldata op het woordniveau. Als laatste werden enkele hiaten in huidige subwoordtokenisers geïllustreerd.

Hoofdstuk 2 gaf een uitvoerige literatuurstudie over de historische en technische context van het BPE-subwoordtokenisatiealgoritme. BPE (Sennrich e.a., 2016) is van oorsprong een compressiealgoritme dat overgezet is naar een populair LSTM-vertaalsysteem (Bahdanau e.a., 2015), en is op die manier overgenomen in transformers (Vaswani e.a., 2017). Belangrijke systemen in het gerelateerde onderzoeksdomein van woordsplitsing bleken op een meetkundig gemiddelde van woordfrequenties gebaseerd (Koehn en Knight, 2003), waarvan de valse positieven grotendeels verholpen konden worden met een morfologische analysator (Fritzinger en Fraser, 2010).

Na een algemene introductie tot statische en contextuele embeddings volgde een bespreking over het combineren van embeddings. Vooreerst werd het verband tussen semantische operaties en vectoroperaties in $\mathbb{L}^2$, gepopulariseerd door Mikolov e.a. (2013b), in vraag gesteld; het wordt tegengesproken door meerdere werken, die vinden dat het gemiddelde van een rij embeddings geen goede embedding voor hun semantische compositie is. Beter zijn complexe, leerbare transformaties zoals een algemene lineaire regressie (Yazdani e.a., 2015), auto-encoder (Salesky e.a., 2018), of BiLSTM (Ács e.a., 2021). De tegenhangers van hen die compositiefuncties leren, zijn zij die componeerbare embeddings leren, met transformerachtigen op kop (Wang e.a., 2021). De conclusie was dat

men het samenstellen liefst zo lang mogelijk uitstelt, en dat het interessanter is om te leren werken met rijen van embeddings.

Tot slot kwam een uitvoerige bespreking van subwoordtokenisers aan bod. Eerst stelde ik een overkoepelende notatie van tokenaantallen voor (Appendix A), en vervolgens stelde ik o.b.v. de literatuur een veralgemeende levenscyclus van subwoordtokenisers voor (§ 2.5.1.2) waar ook complexere tokenisers dan BPE in passen. Het maakt onderscheid tussen het leren van een vocabularium (vocabularisatie), het leren van de segmenteerder die dat vocabularium gebruikt (inferisatie), het segmenteren van het trainingcorpus (corpussegmentatie) en, indien van toepassing, het segmenteren van invoer achteraf (invoersegmentatie).

Vervolgens besprak ik drie grote lijnen van subwoordtokenisers. Tokenisers die zowel in LM's als TM's werken, werden opgedeeld in metriekvarianten van BPE (DLG'-BPE van Wu en Zhao (2018), S-BPE van Vilar en Federico (2021) en WPM van Schuster en Nakajima (2012)), gebruiksvarianten van BPE (BBPE van Radford e.a. (2019) en iBPE van Salesky e.a. (2018)), samplebare tokenisers (ULM van Kudo (2018) en BPE-dropout van Provilkov e.a. (2020)), en de Morfessor-familie (o.a. Baseline Creutz en Lagus (2002), Cat-MAP Creutz en Lagus (2005a), FlatCat Grönroos e.a. (2014), EM+Prune Grönroos e.a. (2020)) en haar derivaten (LMVR van Ataman e.a. (2017) en M-BPE van Banerjee en Bhattacharyya (2018)). Voor ULM (Algoritme 2.3) en Morfessor Baseline (Algoritme 2.4) stelde ik zelf pseudocode op ter formalisatie van wat in de respectieve papers beschreven stond.

Er zijn ook tokenisers die alleen in TM's werken omdat ze specifiek een doeltaalcorpus segmenteren. CSCS-BPE (Huck e.a., 2017) plaatst op voorhand spaties tussen affices en woordstammen opdat BPE ze niet zou samenvoegen, en hanteert een efficiënt vocabularium zonder hoofdletters of spaties in de types om de decoder zoveel mogelijk vrijheid te geven. MorphGen (Tamchyna e.a., 2017) verwerpt daarentegen juist dat de decoder moet leren om tokens aan elkaar te rijgen, en heeft een decoder die louter grammaticale analyses produceert die worden naverwerkt tot woorden via een deterministische regelbank. DPE (He e.a., 2020) benadrukt dat tokenisatie zoveel mogelijk geconditioneerd moet zijn op de beschikbare omgevingsinformatie, met name de tokens van de bronzin en de voorgaande karakters van een de doelzin. Gebruik van karakters als context laat toe om te optimaliseren overheen alle mogelijke segmentaties via een viterbialgoritme.

Hoofdstuk 4 besprak ongeveer 20 gebreken in BPE-tokenisers aan de hand van BPE_{nl}, de BBPE-tokeniser van RobBERT-2020 (Delobelle e.a., 2020). Het hoofdidee van die gebreken werd reeds samengevat onder hun respectieve sectietitel. Ik tracht alleen bijkomende bevindingen te recapituleren.

§4.1: Modellen die meerdere alfabetten willen ondersteunen, moeten ofwel hun vocabularium initialiseren met tienduizenden karakters zoals XLM (Lample en Conneau, 2019), wat reeds de helft van een standaardvocabularium is (§4.12), ofwel met bytes werken, wat dan weer té toelatend is en o.a. voor 500 nonsenstypes in BPE_{nl} zorgt (Tabel D.1).

§4.2: BPE_{nl} bevat ook meer dan 500 types die uit cijfers bestaan, terwijl cijfers helemaal geen morfologie hebben. Getallen die numeriek gelijkaardig zijn worden daardoor mogelijks anders gesegmenteerd, en zijn dan moeilijk te interpreteren door het model.

§4.3: Er zijn vijf stijlen om spaties in subwoorden te coderen (Tabel 4.1): met een apart spatietoken (CSCS-BPE), met een alfabet van SOW-geprefixeerde karakters (BERT-WPM) of EOW-gesuffixeerde karakters (BPE v0.2), of met een los SOW-karakter (BPE_{nl}) of EOW-karakter (BPE v0.1). Ze hebben wel degelijk verschillende effecten: de kern van een Nederlandse samenstelling staat op het einde, dus de losse voorkomens van die kern

prefixeren met een SOW (wat in BPE_{nl} gebeurt) incentiveert een andere segmentatie voor de gebonden en de losse versie (Tabel 4.7). §4.16.3 staat die conclusie bij door te vinden dat 50% van het BPE_{nl}-vocabularium uit subwoorden bestaat die wel beginnen met SOW maar geen variant zonder die SOW hebben in het vocabularium.

§4.4: De standaardgrootte van BPE-vocabularia, $|V| = 30\text{k}$, verhindert dat transformer-decoders leren om morfemen op elkaar af te stemmen, wat nodig is in klinkerharmonie. Vocabularia die dichter bij het karakterniveau zitten, bv. met $|V| = 500$, doen het beter.

§4.5: BPE produceert geen kansverdeling over segmentaties, en hoewel BPE-dropout dat remedieert, is de kansverdeling die BPE-dropout produceert een model voor niets anders dan BPE-dropout zelf. Er zit geen taalkundige en zelfs geen frequentiegebaseerde motivatie achter.

§4.6: Talen zoals het Hebreeuws hebben woordsegmentaties die sterk van de omringende context afhangen, maar BPE segmenteert woorden in een vacuüm.

§4.7: Dat BPE zijn invoer segmenteert door de leerfase opnieuw af te spelen, modelleert zélfs het compressieproces waarop BPE gebaseerd is niet goed. Daarbij gaat maximale compressie in tegen morfologische splitsing, die juist wel meerdere tokens wil overhouden.

§4.8: In meertalige toepassingen domineren de populairste alfabetten het vocabularium. Webcorpora zoals dat waarop BPE_{nl} getraind is, zorgen bovendien voor een overrepresentatie van types in V (minstens 180) die louter met pornografie te maken hebben (Tabel D.2).

§4.9: Afhankelijk van het domein waarop BPE getraind wordt, verschilt tot wel 60% van het vocabularium. Een BPE-tokeniser uitbreiden door de V en M van een andere BPE-tokeniser achteraf toe te voegen (zoals in RobBERT-2022), heeft niet het verwachte effect (Figuur D.11).

§4.10: De aanwezigheid van "stapsteentypes" in het BPE-vocabularium wordt bevestigd door de fractie van types die het wel tot in het vocabularium hebben geschopt (wat een hoge frequentie vergt) en toch nooit als token voorkomen in het corpus na afloop van de leerfase. Hun bestaan wordt ook bevestigd door een NMT-decoder gradueel meer types te laten gebruiken (iBPE), waarbij hij tot 25% van alle types vergeet te gebruiken.

§4.11: De onderliggende strings van BPE-types komen minder voor als tokens dan als strings in het corpus voor en na segmentatie, en sterker nog, 32% van alle mogelijke typeparen verwisselt van volgorde wanneer we sorteren op tokenaantal t.o.v. wanneer we sorteren op stringaantal. De verhouding tussen stringaantal en tokenaantal is het grootst voor types die zo lang zijn als morfen (vergelijk Figuur D.1 en Figuur D.14). Een manier om dergelijk dataverlies tegen te gaan, is om embeddings die elkaars data stelen met elkaar te laten communiceren, zoals in iBPE.

§4.12: De maximale vocabulariumgrootte $|V| = \tau$ is een hyperparameter die klakkeloos is overgenomen van de ene paper op de andere, zelfs met de overgang van LSTM's naar transformers, terwijl er voldoende bewijs is dat die twee architecturen zich anders gedragen. Transformers werken beter met kleinere τ; dat niet doen riskeert om het gros van het parameterbudget aan statische embeddings te spenderen (tot wel 71%). Er zijn verschillende automatische methodes om τ te vinden, maar ze botsen altijd op dezelfde paradox in BPE: τ moet klein zijn opdat BPE morfemen geen onderdeel van grotere tokens maakt, maar τ moet groot zijn vooraleer BPE non-compositionele woorden in V kan opnemen. Anderzijds wijst lexicalisatie bij grote τ op overfitting, waardoor de tokeniser domeinafhankelijk wordt.

§4.13: Morfemen die klein genoeg zijn, verdampen ingeval de tokeniser ze in tweeën splitst. Een morfologisch corpus als e-Lex (§C.2) bevat referentiesplitsingspunten waartegen we splitsingen van een tokeniser kunnen vergelijken als een binair classificatieprobleem. Ik stel een nieuw framework op om de impact van valse positieven en valse negatieven kwalitatief te analyseren. De analyse van een fout bestaat uit de verklaring van wat de fout is (hetgeen helpt bij het produceren van voorbeelden), welke merge er schuldig is voor de fout (hetgeen terugkomt in §5.3), of de fout een geval van aliasing is (morfemen die per abuis herkend worden in andere morfemen, en omgekeerd), en of de fout ernstig is (zoals het samensmelten van samenstellingen). Ik verwijs naar §B.4 voor de implementatie van het algoritme dat morfologische splitsingen afleidt uit e-Lex.

§4.14: Wanneer een morfeem verschijnt naast een string die er een ander morfeem mee kan vormen, kan BPE de nodige nuance niet brengen. Met een prefixboom wordt een analyse gemaakt van morfen die andere morfen inleiden en bij segmentatie foutief verschijnen (positieve aliasen), en van morfen die gevaar lopen om foutief in grotere morfemen (negatieve aliasen) opgenomen te worden. 60% van alle morfen loopt gevaar op positief gealiast te worden, en 10% daarvan materialiseert. 7% van alle morfen loopt gevaar op negatief gealiast te worden, en dit keer komt dat in 34% van de gevallen uit.

§4.15: Typo's van levenshteinafstand 1, alsook karakterverwisselingen, zorgen voor nieuwe BPE-splitsingsposities waarvan amper de helft voorkomt zonder de typo's (Tabel 4.4).

§4.16: BPE ziet de grens tussen de deelwoorden van een samenstelling als bewijsmateriaal om de desbetreffende merge uit te voeren, terwijl het juist omgekeerd moet zijn. BPE merget de grenzen van samenstellingen doorheen de hele leerfase, reeds vanaf de eerste aangeleerde merges, waardoor de segmentaties van de deelwoorden meteen onherkenbaar worden en er dataverlies optreedt. Sommige letters hebben een hoge affiniteit om in één richting te mergen (Figuur D.18), wat tot systematische fouten in de segmentaties van samenstellingen leidt (Tabel D.4). Interfices zorgen eveneens voor systematische fouten, gezien 95% van alle interfix-*s*'en mee opgenomen wordt in het token links of rechts ervan (Tabel 4.6).

§4.17: Ik bewijs een invariant in BPE-tokenisatie, de functionele erfzonde, die stelt dat elk subwoordtype slechts op één manier gemerged kan worden, mede doordat elke merge overal in het corpus tegelijk gebeurt. Veel van de bovenstaande problemen hebben een link met die invariant.

Hoofdstuk 5 onderzoekt tot slot twee richtingen om de segmentatiestap van BPE te verbeteren. Enerzijds worden 12 verschillende vooraf getrainde vocabularia gecombineerd met 7 nieuwe segmentatiemethodes. De vocabularia zijn deels substringverzamelingen uit e-Lex en deels BPE-vocabularia, en van de segmentatiemethodes zijn er 4 gebaseerd op shift-reducers en 2 zijn random-access. Van alle methodes bleek een random-access-greedy segmentatie het best te werken.
Anderzijds werd een nieuwe techniek voorgesteld om de mergegraaf van BPE achteraf te bewerken door bepaalde types uit de graaf te slaan (knockout). Daarbij wordt elk type door zijn ouders vervangen in de merges waarin het deelneemt. Een aanpassing aan BPE laat toe om tuples van tokens, niet alleen paren, in een merge te betrekken (BTE). Beslissen welke types te verwijderen gebeurt a.d.h.v. de fractie valse negatieven die de overeenkomstige merge overheen zijn toepassingen in e-Lex produceert. BPE-knockout presteert beter op morfologische splitsingen dan BPE, zelfs ingeval slechts een klein deel van de knockouts die in aanmerking komen ook echt gebeurt. Een complementaire stap op basis van valse positieven (annealing) toonde minder effect.

6.2 Conclusie

Hoofdstuk 3 stelde twee onderzoeksvragen voorop: Onderzoeksvraag 1 peilde naar de problemen met BPE-tokenisers, waarvoor hierboven heel wat bewijs gegeven werd. De conclusie is duidelijk: de BPE-tokeniser heeft fundamentele, onoplosbare gebreken die veroorzaakt worden door zijn roots als hiërarchisch compressiealgoritme, hetgeen des te duidelijker wordt naarmate de gesegmenteerde taal rijkere morfologie bevat (met name agglutinatie) of een groot, niet-fonologisch alfabet heeft.

De toekomst ligt in generatieve subwoordtokenisers (met name die waarin de segmentatie van elk woord geoptimaliseerd wordt met een viterbialgoritme) en in tokenloze end-to-end-modellen, die mettertijd dieper en goedkoper zullen worden. Voor die modellen die reeds op een BPE-tokeniser getraind zijn, peilde Onderzoeksvraag 2 naar remedies vertrekkende vanuit zo'n bestaande tokeniser. Semi-supervisie op basis van morfologische kennis, voorgesteld als BPE-knockout, toont dat er effectief remediëring mogelijk is.

6.3 *Future work*

Hoewel ik initieel vreesde om zonder ideeën te vallen, ben ik na dit thesisonderzoek met veel meer vragen geëindigd dan ik begonnen ben. Hoewel enkele uitgevoerde experimenten en geproduceerde grafieken het reeds niet tot in de thesis gehaald hebben, zijn er alsnog extra experimenten die ik met meer tijd had willen uitvoeren. Hieronder een lijst van interessante onderzoeksrichtingen die sommige van die vragen zouden beantwoorden.

6.3.1 Over tokenisers

- Ik focuste mij in mijn experimenten op intrinsieke evaluatie m.b.v. morfologische splitsingspunten in e-Lex. Volgens de MH in Hoofdstuk 1 impliceert een betere intrinsieke evaluatie ook een betere extrinsieke evaluatie, maar dat moet geverifieerd worden: welk van de tokenisers in §2.5 zorgt voor het best presterende LM? Er is nood aan een grote vergelijking die bv. één RoBERTa traint voor elke tokeniser.
- Hoewel Morfessor een oud algoritme is, kent zijn familie al 20 jaar innovaties. In Hoofdstuk 4 zagen we dat Morfessor veel van de problemen met BPE kon ontwijken. Toch heeft het geen implementatie in het `tokenizers`-pakket van HuggingFace, dat alleen BPE, WPM en ULM aanbiedt, en voor zover ik weet zijn er geen recente papers die Morfessor met BPE vergelijken buiten die over Morfessor zelf. Ik pleit voor het mainstreamen van de Morfessor-familie.
- End-to-end-modellen zijn reeds veelbelovend, zelfs al zijn ze pas de laatste jaren opgekomen. Heeft subwoordtokenisatie nog een toekomst, of moet al onze belangstelling naar end-to-end-modellen uitgaan? Er is nood aan een grote vergelijking tussen de beste subwoordtokenisers (bv. ULM en Morfessor) en de beste karaktertokenisers (bv. Charformer).
- Kunnen de paradigma's van stringbewerkingen en neurale bewerkingen gecombineerd worden?
- Is het beter om zoals CSCS-BPE een apart token te gebruiken voor spaties, of hebben modellen alsnog baat bij het opnemen van een SOW of EOW in subwoorden? Welk van de stijlen in Tabel 4.1 is beter? Welke aspecten van de taal in kwestie beïnvloeden de conclusie?

- Is het mogelijk om een viterbitokeniser voor LM's op te stellen die kansen gebruikt met op zijn minst een beetje bidirectionele conditionering, i.p.v. de unigram-aanname te maken zoals ULM?

6.3.2 Over metrieken

- Zijn er betere metrieken voor BPE dan die van S-BPE? Kunnen we statistieken i.v.m. het respecteren van morfologische grenzen, met name $R(m)$ en $B(m)$ uit §5.3 – gemeten op e-Lex-types of op e-Lex-tokens in OSCAR – rechtstreeks als BPE-metriek gebruiken, in plaats van tokenpaarfrequentie?
- De frequentste merge krijgt mogelijks een significant deel van zijn frequentie van valse negatieven (§ 4.14). Hoge frequentie slaat in dat geval op twijfel, niet op zekerheid. Het kan beter zijn om eerst mínder frequente merges uit te voeren. Kunnen we morfologische statistieken gebruiken als een regularisatieterm bij de BPE-metriek?
- Met andere woorden is de volgorde van de mergelijst M zoals BPE ze opstelt niet de meest optimale. Is er een intelligente manier om M te herordenen? Kan dat met een genetisch algoritme?
- OFS is een invariant van zowel BPE als BPE-knockout. Is er een modificatie aan BPE mogelijk die OFS oplost? Is er bijvoorbeeld een manier om BPE-merges tijdens het trainen "zacht" uit te voeren, opdat de versie van elke string voor en na de merge aanwezig blijft in het corpus (met een zeker totaal gewicht) en foutieve merges minder permanentie hebben?
- BPE-knockout kiest welke merges moeten verdwijnen op basis van blame. Net als BPE kunnen we dat echter veralgemenen naar eender welke metriek die een merge op een getal afbeeldt. Zijn er andere metrieken die betere keuzes maken? Kan BPE-knockout bijvoorbeeld de interfixfouten in Tabel 4.6 oplossen door zich daarop te focussen? Kan BPE-knockout datainversie oplossen door gebruik te maken van de inversieverhouding (§4.11)?
- Is er een eenvoudige manier om BPE afhankelijk te maken van de PoS van een woord, hetgeen voordelig bleek in de woordtokenisers van § 2.4.1?

6.3.3 Over embeddings

- Kan men op basis van embeddingsimilariteit (rekening houdend met anisotropie van Ethayarajh (2019) en Luo (2021)) meerdere types aan dezelfde geclusterde embedding toewijzen, en zo de embedding-LUT verkleinen?
- Zoals aangehaald in §4.11, kunnen subwoorden er baat bij hebben om kennis met elkaar te communiceren. Tijdens iBPE geven ouders kennis door aan hun kinderen. Om inversie tegen te gaan, zou het andersom moeten zijn, en voorts weten we dat er andere paden in de BPE-mergegraaf zijn waarlangs informatie delen waardevol is (bv. tussen *coronamaatregelen* en *corona* in Figuur 1.2, waar *corona* niet de ouder maar de halfbroer van *coronamaatregelen* is en er dus een pad van twee stappen gevolgd moet worden). Hoe bepalen we die paden? En indien we ze kennen, hoe wisselen we informatie het best uit tussen subwoorden?
- Hoeveel van de kennis van een transformer zit in de contextualiserende modules, gegeven dat een significante fractie van alle gewichten in statische embeddings geïnvesteerd zit?

- Zijn de statische embeddings van een transformer eerder een $\mathbb{L}^2$-representatie van strings, of eerder van hun betekenissen?

What's in a token?

Deel III

Appendices

A Notatie

Deze appendix licht de notatie gebruikt voor wiskundige formules in Hoofdstuk 2 toe. Met name is het nodig om symbolen te hebben voor de verschillende soorten aantallen die men kan tellen in een corpus.

Hoewel het corpus uit zinnen bestaat die in subwoordtokens worden gesplitst, is het alsnog nuttig om ook notatie te hebben voor woorden, gezien BPE-tokenisers woordgrenzen respecteren en daarvan gebruik maken in hun implementaties. Quasi alle talen zetten spaties tussen hun woorden, dus is het triviaal om ze te herkennen met een pre-tokeniser.

In al het onderstaande is "frequentie" gelijk aan "aantal voorkomens (tokens)".

$t_1 t_2$	concatenatie van twee types t_1 en t_2.
$\mathcal{D}$	lijst van alle zinnen (inclusief duplicaten) in het corpus.
$\mathcal{T}_{\mathcal{D},\theta}$	lijst van alle tokens (inclusief duplicaten) in het corpus $\mathcal{D}$ gesegmenteerd met tokeniser θ.
V_θ	verzameling van alle subwoordtypes van de tokeniser.
$W_\mathcal{D}$	verzameling van alle woordtypes in corpus $\mathcal{D}$.
$C_\mathcal{D}(w)$	frequentie van woord $w \in W$ in corpus $\mathcal{D}$.
$C_{\mathcal{D},\theta}(t)$	frequentie van type $t \in V$ in corpus $\mathcal{D}$ gesegmenteerd met tokeniser θ.
$C_{\mathcal{D},\theta}(t_1, t_2)$	frequentie van de opeenvolging van types $t_1, t_2 \in V$ in het corpus $\mathcal{D}$ gesegmenteerd met tokeniser θ.
$C_\theta(t, w)$	frequentie van type $t \in V$ in woord $w \in W$ gesegmenteerd volgens tokeniser θ.

Merk op dat o.a. de volgende relaties gelden (deels uit Vilar en Federico, 2021):

$$C_{\mathcal{D},\theta}(t) = \sum_{w \in W_\mathcal{D}} C_\mathcal{D}(w) C_\theta(t, w) = \sum_{t' \in \mathcal{T}_{\mathcal{D},\theta}} \mathbb{1}\{t' = t\} \tag{A.1}$$

$$C_{\mathcal{D},\theta}(t) = \sum_{t' \in V_\theta \cup \{\varnothing\}} C_{\mathcal{D},\theta}(t, t') = \sum_{t' \in V_\theta \cup \{\varnothing\}} C_{\mathcal{D},\theta}(t', t) \tag{A.2}$$

$$C_{\mathcal{D},\theta}(t_1, t_2) = C_{\mathcal{D},\theta \cup \{t_1 t_2\}}(t_1 t_2) \tag{A.3}$$

$$|\mathcal{T}_{\mathcal{D},\theta}| = \sum_{t \in V_\theta} C_{\mathcal{D},\theta}(t) = \sum_{t \in \mathcal{T}_{\mathcal{D},\theta}} 1 \tag{A.4}$$

... waarbij $\cup$ een uitbreiding van het vocabularium van θ aangeeft. Wanneer de tokeniser verandert overheen vocabularisatie-/inferisatie-iteraties krijgt θ een subscript, bv. θ_k.

B Viterbialgoritmes

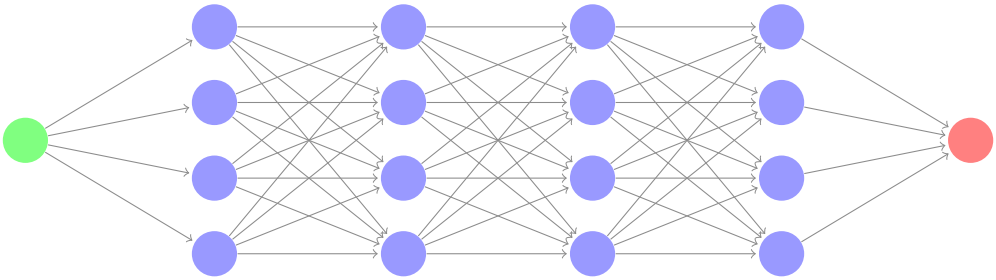
Deze appendix dient om de hoofdgedachte achter een *viterbialgoritme* uit te leggen, voor wie een opfrisser nodig heeft. Viterbialgoritmes zijn een vorm van *dynamic programming*, d.w.z. algoritmes die een optimale oplossing op een probleem vinden door iteratief of recursief hetzelfde probleem op kleinere schaal op te lossen, en daarbij nooit werk onnodig herhalen. Ze zijn vernoemd naar een idee van Viterbi (1967) in signaalverwerking.

Een viterbialgoritme geeft vaak de indruk dat het greedy – en dus naïef en suboptimaal – is, terwijl het wel degelijk een globaal optimum vindt. Vandaar is het belangrijk om de sleutelgedachte te begrijpen. Ik illustreer het principe eerst m.b.v. pathfinding, en pas het daarna toe op de segmentatie van woorden in een unigram-LM zoals dat van Kudo (2018). Mijn toepassing op pathfinding is een abstractie van het viterbialgoritme dat DeRose (1988) introduceerde om PoS-tagging met HMM's in kwadratische i.p.v. exponentiële tijd uit te voeren.

B.1 Viterbi voor pathfinding

Stel, we hebben een graaf die bestaat uit n lagen met elk H vertices, waarbij elk opeenvolgend paar van lagen een volledige bipartiete graaf is (d.w.z. alle vertices van laag i zijn verbonden met alle vertices van laag $i+1$). Elke verbinding (*edge*) tussen twee vertices (v_i, v_j) draagt een zekere score $w(v_i, v_j)$.

Laat ons ook een begin- en eindvertex toevoegen, waarvan de edges allemaal score 1 hebben.



Noteer de i'de vertex van laag ℓ als $v_{\ell,i}$, en een pad als $\pi = (v_*, v_{1,i_1}, v_{2,i_2} \dots v_{n,i_n}, v^*)$. Het doel is nu om het pad π^* van de beginvertex v_* naar de eindvertex v^* te vinden dat

over een maximaal scoreproduct stapt:

$$\pi^* = \arg\max_{\pi} W(\pi) \qquad \text{met } W(\pi) = \prod_{\ell=1}^{n} w(\pi[\ell-1], \pi[\ell]) \tag{B.1}$$

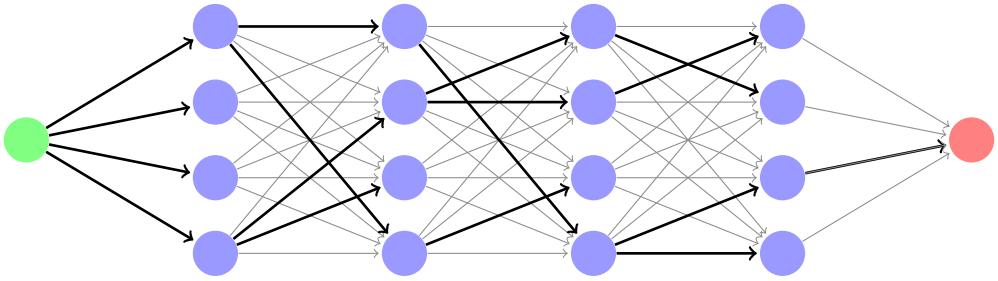
Er zijn H^n mogelijke paden waartussen we kunnen kiezen, dus ze gewoon allemaal enumereren kost $O(H^n)$ tijd. Dat hoeft echter niet om het globaal optimum te vinden, waar een viterbialgoritme gebruik van maakt om hetzelfde resultaat op slechts $O(H^2 n)$ tijd te vinden.

Een belangrijk inzicht is dat *als* de i'de vertex van laag ℓ, genoteerd $v_{\ell,i}$, op het optimale pad tussen v_* en v^* ligt, dan *moet* het deel van dat pad dat tussen v_* en $v_{\ell,i}$ loopt, ook het optimale pad zijn tussen v_* en $v_{\ell,i}$. Immers, als dat niet zo was, dan bestond er een betere manier om tot bij $v_{\ell,i}$ te geraken zonder dat het pad van $v_{\ell,i}$ naar v^* zou veranderen, en dat schendt de optimaliteit van het volledige pad. Het feit dat ons optimalisatieprobleem bestaat uit 1. kleinere varianten van zichzelf bestaat waarvan 2. de oplossing onafhankelijk is van de rest van het grotere probleem, heet *optimale substructuur*.

Stel nu dat we het probleem opgelost hebben voor alle vertices in de voorlaatste laag, waarbij elke vertex $v_{n-1,i}$ zowel het beste pad $\pi^*_{n-1,i} = (v_*, \ldots, v_{n-1,i})$ met zich meedraagt als de score $W(\pi^*_{n-1,i})$. Een vertex $v_{n,j}$ in de laatste laag moet dan enkel nog kiezen welk van de H vertices in de vorige laag voor de hoogste score zorgt, rekening houdend met niet alleen $W(\pi^*_{n-1,i})$ maar ook de extra score die springen naar $v_{n,j}$ opbrengt. Voor elke $j = 1 \ldots H$ zoeken we m.a.w.

$$W(\pi^*_{n,j}) = W(\pi^*_{n-1,i} \cup v_{n,j}) = \max_i \{w(v_{n-1,i}, v_{n,j}) W(\pi^*_{n-1,i})\}. \tag{B.2}$$

Die redenering kunnen we herhalen voor de keuze die in de voorlaatste laag overheen de laag daarvoor werd gemaakt om tot $W(\pi^*_{n-1,i})$ te komen, en zo redeneren we steeds achteruit tot we uiteindelijk bij het triviale geval van $W(\{\}) = 1$ komen. Merk op dat er in elke laag H vertices zijn die naar de H vorige vertices kijken, en er n lagen zijn; vandaar de tijdscomplexiteit $O(H^2 n)$.



Maar is zo'n lokale keuze dan niet greedy? Neen! De bovenstaande keuze maximaliseert de *cumulatieve* score over het pad. Een greedy keuze vertrekt daarentegen in de voorlaatste laag, en probeert alleen de *bijkomende* score te maximaliseren. Voor elke $i = 1 \ldots H$ zoeken we dan

$$\max_j w(v_{n-1,i}, v_{n,j}). \tag{B.3}$$

Om het nog een laatste keer te benadrukken:

Greedy: kijk voorwaarts en kies de stap naar daar met de hoogste score.
Viterbi: kijk achterwaarts en kies de stap naar hier met de hoogste combinatie van geaccumuleerde score en score om te stappen.

Noteer tot slot dat als de scores kansen zijn ($\in [0,1]$) een product numeriek instabiel is. Aangezien $\ln(\max_x f(x)) = \max_x \ln f(x)$, kan men overal de logaritme nemen om numerieke stabielen sommen i.p.v. producten te krijgen ($\ln \prod_x f(x) = \sum_x \ln f(x)$). Het gebeurt uitzonderlijk dat men een viterbialgoritme voor een som van producten probeert op te stellen, maar dat zorgt voor problemen gezien de logaritme niet door een som kan bewegen en het product dus niet kan stabilisren.

B.2 Viterbi voor stringsegmentatie

Stel dat we een string s van karakters $c_1 \dots c_n$ hebben. De karakters kunnen worden gegroepeerd tot tokens op 2^{n-1} verschillende manieren (cfr. § 2.4.1). Stel dat elk token t een score $P(t)$ krijgt, en de score van een sequentie van tokens hun scoreproduct is.

De vraag is nu om een sequentie tokens te vinden met maximaal scoreproduct *en* samen s vormen indien geconcateneerd. Er is duidelijk een gelijkenis met de vorige sectie. Het inzicht in dit geval is dat *als* een sequentie van tokens de volledige optimale tokensequentie van s inleidt, dan *moet* die sequentie de optimale tokensequentie zijn die de overeenkomstige substring segmenteert. Dat is optimale substructuur.

De graaf is in dit geval echter niet mooi in bipartiete lagen opgedeeld, al zijn er n karakters. Beschouw de string $s =$ *loremipsum*. Het laatste token in die string zou *m* kunnen zijn, en dan moeten we eerst het probleem voor de substring *loremipsu* oplossen. Het laatste token zou ook *um* kunnen zijn, en dan moeten we alleen *loremips* oplossen. Sommige paden (tokensequenties) kunnen dus korter zijn dan andere – het langste pad is n tokens lang (elk een karakter), het kortste pad is 1 token lang (de hele string).

Een string van lengte n kijkt m.a.w. naar alle beëindigende tokens $c_{i+1} \dots c_n \in V$ en de beste score van de string voordien:

$$W(\pi_n^*) = \max_{i=0..n-1} \{P(c_{i+1} \dots c_n)\, W(\pi_i^*)\} \tag{B.4}$$

waarbij π_i^* de beste tokenisatie van de substring $c_1 \dots c_i$ is, en $c_{i+1} \dots c_n \in V$ een token in het vocabularium. Er zijn n strings waarvoor we W moeten evalueren, waarbij een string van lengte i over i voorgangers redeneert: dat heeft dan complexiteit

$$\sum_{i=1}^{n} i = O(n^2) \tag{B.5}$$

wat heel wat beter is dan $O(2^n)$.

B.3 Implementatiestijl

Het ligt voor de hand om een viterbialgoritme recursief te implementeren en van achter naar voren te werken, steeds kleinere problemen oplossend. Er is een tweede populaire manier, die de hele tijd in dezelfde scope werkt zonder functie-oproepingen: van voor naar achter, in een dubbele genestelde lus.

Hierboven schreef ik steeds dat latere vertices $v_{\ell,i}$ (of strings $c_1 \dots c_\ell$) "kijken" naar eerdere vertices $v_{\ell-1,i}$ (of strings $c_1 \dots c_i$) en "beslissen" welke de beste voorganger is. De voorwaartse manier draait de rollen om: de vroege vertices bieden zichzelf aan elke mogelijke opvolger aan, en de opvolger heeft een geheugen van de beste aanbieding tot dan toe. Wanneer de vroegste vertices al hun aanbiedingen hebben gedaan, kunnen we naar de tweede laag gaan: aangezien de vertices in die laag normaliter alleen naar die in

de eerste laag kijken, hebben ze nu alle mogelijke voorgangers al gezien, en is hun huidige beste voorganger effectief de optimale. Eens de tweede laag zich heeft aangeboden aan al zijn opvolgers, gaan we naar de derde laag, die normaal zou kijken naar de eerste en tweede laag, en die dus weer reeds allemaal aan bod zijn gekomen.

Algoritme 2.3 is zo'n voorwaarts viterbialgoritme voor stringsegmentatie in het unigram-LM van Kudo (2018).

De unigram-aanname, i.e. dat de tokenkansen (hierboven "scores") onafhankelijk zijn van de omringende tokens, is strikter dan nodig. Zolang het probleem zijn optimale substructuur behoudt, d.w.z. dat het kleinere probleem niet afhangt van wat er in het grotere probleem bijkomt, kunnen we een viterbialgoritme gebruiken om niet alle paden te bekijken. He e.a. (2020) stellen daarom een (voorwaarts) algoritme voor dat als gewicht niet $P(t)$ gebruikt, maar $P(t \mid c_1 \dots c_i)$. *Let op:* $c_1 \dots c_i$ is uitdrukkelijk een string en geen sequentie van voordien gekozen tokens, want stel nu dat er een heel laag scorend pad van c_1 naar c_i uitgebreid kan worden met een heel hoge score terwijl een hoog scorend pad van c_1 naar c_i alleen uitbreidingen met lage scores heeft, dan is het onmogelijk om op voorhand te weten welk pad nu voor de uiteindelijke optimale oplossing zorgt – er is dan geen optimale substructuur. Vandaar dus dat het uitbreiden vanaf c_i niet mag afhangen van de gekozen tokens, en alleen van $c_1 \dots c_i$ zelf (hetgeen gelijk is voor eender welke sequentie van tokens die die string bedekt).

B.4 Viterbi voor morfeemalignering

Zoals elders uitgelegd, bevat de CELEX-dataset morfologische ontbindingen van woorden, maar zonder er rechtstreekse segmentaties van te zijn: morfemen zijn onderliggende, abstracte, theoretische atomen, die zich aan de oppervlakte als morfen voordoen die allerlei vervormingen van de morfemen kunnen vertonen. Zo ontbindt CELEX de samenstelling *kolencentrale* in de morfemen *kool en centrum aal e*, terwijl we eigenlijk de morfen *kol en centr al e* willen. De twee zijn duidelijk gerelateerd:

morfemen:	kool	en	centrum	aal	e
morfen:	kol	en	centr	al	e

Merk op dat morfemen soms met een leeg morf overeenkomen. Voor *acceptatiegraad*:

morfemen:	accept	eer	atie	graad
morfen:	accept		atie	graad

Hoe gaan we van een woord w en zijn CELEX-morfemen $[m_1 \dots m_M]$ naar zijn morfen? Laat ons aannemen dat woorden als volgt gesynthetiseerd worden uit de lijst van morfemen:

1. Voor elk morfeem m: beslis of m verdwijnt of niet.
2. Voor elk morfeem m dat overblijft: kies eender welk aantal karakters tussen 0 en $|m| - 1$ om te laten vallen *van het einde* van m.
3. Beginnend bij een lege string s: voor elk morfeem m, concateneer een willekeurige string van 0 of meer karakters aan s, en vervolgens m.
4. Eindig met een willekeurige string van 0 of meer karakters.

We zoeken m.a.w. dan naar een zo lang mogelijke *prefix* van zo veel mogelijk morfemen, en in volgorde. De willekeurige strings die zich tussen die prefices bevinden, worden erachter geplakt. Een eerste idee is een greedy algoritme dat door de lijst van morfemen itereert en telkens naar de eerste letter zoekt. Indien niet gevonden, herbegint het aan

het begin van de huidige string. Indien wel gevonden, plakt het de gepasseerde substring aan het vorige morf, en consumeert het een zo groot mogelijke prefix, om dan van daar verder te itereren. Voor *kolencentrale*:

morfeem	**invoer**	**uitvoer**
kool	kolencentrale	
centrum	lencentrale	ko
aal	ale	ko+len centr
e	le	ko+len centr a
		ko+len centr a+l e

In dat geval werkt het goed. Kijk echter wat er gebeurt voor *acceptatiegraad*:

morfeem	**invoer**	**uitvoer**
accept	acceptatiegraad	
eer	atiegraad	accept
atie	graad	accept+ati e
graad	ad	accept+ati e+gr a
	ad	accept+ati e+gr a
		accept+ati e+gr a+ad

Uiteraard is *acceptati egr aad* fout. Het probleem is zit niet alleen in overbodige morfemen:

morfeem	**invoer**	**uitvoer**
isoleer	isolementspositie	
ement	mentspositie	isole
s	ntspositie	isole+m e
pose	positie	isole+m e+nt s
eer	itie	isole+m e+nt s pos
itie		isole+m e+nt s pos+iti e
		isole+m e+nt s pos+iti e

Ook *isolem ent s positi e* is niet correct. De greedy aanpak wordt om de tuin geleid door letters die uit een morfeem zijn weggevallen, van andere morfemen te stelen: *isoleer* heeft als morf *isol*, en de *e* erachter hoort eigenlijk bij het volgende morfeem.

Het had uiteraard ook omgekeerd kunnen zijn. Er is dus een aanpak nodig die de twee hypotheses test; de winnende hypothese is diegene waarvoor de meeste letters een morfeem prefixeren. Hierboven is het bv. superieur om de prefices *isol* (4 letters) en *ement* (5 letters) te herkennen dan *isole* (5 letters) en *e* (1 letter).

Eén zo'n aanpak is een bruteforce-algoritme dat dat getal bepaalt voor elke mogelijke segmentatie van de string en elke mogelijke alignering van morfemen met substrings waarbij elk morfeem wel of niet toegewezen wordt en waarbij de volgorde van toewijzingen de volgorde van morfemen moet gehoorzamen. Het equivalente combinatorische probleem heeft grootte-orde $O(|w|^{|w|})$, dus is de bruteforce-aanpak praktisch onmogelijk.

We zagen in §B.2 een stringsegmentatieviterbialgoritme waarbij elke prefix van w een knoop in de graaf was en het vocabularium V alle toegelaten uitbreidingen bevatte. Het algoritme maximaliseerde het product van de kansen van gebruikte tokens, hetgeen we kunnen vervangen door een som van prefixscores, en V kunnen we dan vervangen door de lijst van morfemen is. Er is echter een groot probleem: V moet statisch zijn opdat er optimale substructuur is. Eens we een morfeem gebruikt hebben om een stap te zetten naar een grotere string, mogen we dat morfeem niet nog eens gebruiken. Dat betekent dat we later in de string niet dezelfde stappen kunnen zetten afhankelijk van

de oplossing vroeger in de string: een schending van optimale substructuur.

Er is een andere graaf die wél optimale substructuur heeft. Bemerk dat de bovenstaande greedy uitwerkingen op iteratie i ofwel morfeem m_i vinden, ofwel morfeem m_i laten vallen, en dat we op dat moment zeker weten dat $m_1 \dots m_{i-1}$ geconsumeerd zijn en $m_{i+1} \dots m_M$ nog niet. Neem nu een graaf waarin een knoop bestaat uit een prefix van w én de index van het huidige te gebruiken morfeem m_i. Die graaf heeft i.t.t de bovenstaande wel optimale substructuur: wat er ook met $m_1 \dots m_{i-1}$ gebeurd is, we weten dat we nog over m_i moeten beslissen, en wat we ook beslissen, dat in de volgende stap de beslissing over $m_{i+1} \dots m_M$ gevrijwaard is.

We kunnen de knopen in een tabel (een *viterbitrellis*) van $(M+1) \times (|w|+1)$ cellen schikken. Voor *isolementspositie* ziet het viterbipad door de tabel eruit als volgt:

		i	s	o	l	e	m	e	n	t	s	p	o	s	i	t	i	e
isoleer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ement	0	1	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5
s	0	1	2	3	4	5	6	7	8	9	9	9	9	9	9	9	9	9
pose	0	1	2	3	4	5	6	7	8	9	10	10	10	10	10	10	10	10
eer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	13	13	13	13
itie	0	1	2	3	4	5	6	7	8	9	10	11	12	13	13	13	13	14
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

De pijlen wijzen van het einde van morf i naar het einde van morf $i-1$. Inderdaad, het viterbipad spelt hier de segmentatie *isol ement s pos itie* uit. Voor *kolencentrale* krijgen we ook het juiste resultaat:

		k	o	l	e	n	c	e	n	t	r	a	l	e
kool	0	0	0	0	0	0	0	0	0	0	0	0	0	0
en	0	1	2	2	2	2	2	2	2	2	2	2	2	2
centrum	0	1	2	2	3	4	4	4	4	4	4	4	4	4
aal	0	1	2	2	3	4	5	6	7	8	9	9	9	9
e	0	1	2	2	3	4	5	6	7	8	9	10	10	10
	0	1	2	2	3	4	5	6	7	8	9	10	10	11

Onderstaande Python-code implementeert het voorwaarts viterbialgoritme dat eerst het trellis genereert en vervolgens de segmentatie afleidt. Een pad doorheen het trellis begint linksboven. Op rij i en kolom j kan het ofwel horizontaal naar kolom $j+k$ bewegen ($k \geq 1$ karakters overslaan, wat niet meetelt bij de score; zie *ol* en *al*) ofwel één rij naar beneden bewegen om morfeem i te consumeren. Als de langste prefix van morfeem i die begint op karakter j in totaal K karakters heeft, dan kan het pad naar elk karakter $j+k$ bewegen voor $k \in \{0 \dots K\}$. Een onzichtbaar morfeem zal in de optimale segmentatie $K = 0$ hebben, en een verticale beweging uitlokken (zie *eer*).

Het algoritme doorloopt elk van de $(M+1)(|w|+1) \approx M|w|$ cellen eenmalig. Voor de

cel op rij i en kolom j wordt elk van de $|m_i|+1$ bewegingen naar rij $i+1$ getest, alsook elk van de $|w|-j$ horizontale bewegingen.

Laat $L = \max_i |m_i|$ de lengte van het grootste CELEX-morfeem zijn, dan is een bovengrens op de kost van het algoritme $O(M|w|(|w|+L))$. Aangezien L niet groter kan zijn dan $|w|$, is $O(|w|^2 M)$ een goede benadering. In de praktijk aligneert het alle 96 540 morfologieën in e-Lex aan ∼7500 woorden/s, d.w.z. dat het de hele dataset op 13 seconden verwerkt. Heel wat beter dan een bruteforce die voor één woord al langer duurt.

Algoritme B.1 Viterbialgoritme voor optimale alignering van morfemen en morfen

```python
@dataclass
class ViterbiNode:
    best_count: int = -1  # Ensures that the backpointer is initialised by the first node that talks to this node.
    backpointer: Tuple[int, int] = None


def morphSplit_viterbi(lemma: str, morphemes: str):
    """
    Splits into morphs rather than morphemes. That is: removing spaces from the result will produce the lemma.
    For the lemma "kolencentrale":
        morphemeSplit: kool en centrum aal e
        morphSplit:    kol en centr al e

    Implementation details:

    Iterative Viterbi algorithm with the same optimal results as a recursive bruteforce, except the problem goes
    from completely intractable (several minutes, running out of memory) to trivial (0 seconds and a small table).

    Viterbi here is NOT as simple as a flat string segmentation. The reason is that the allowed vocabulary
    (the set of steps to new nodes) CHANGES depending on which steps have been made on the path to a
    node, meaning you can't be sure which solution is optimal up to that node without knowing what happens after.

    Here's how I re-interpreted the problem to Viterbi: instead of a substring by itself being a node in
    the search graph, a node is a pair of (substring, available vocab), where 'available vocab' is the start of the
    sublist of morphemes left out of all morphemes available.
    """
    lemma_normed     = normalizer.normalize_str(lemma).lower()
    morphemes_normed = normalizer.normalize_str(morphemes).lower()

    morpheme_prefices = [[morpheme[:i] for i in range(len(morpheme) + 1)] for morpheme in
                         morphemes_normed.split(" ")]
    n_morphemes = len(morpheme_prefices)
    n_chars     = len(lemma_normed)
    n_rows_trellis = n_morphemes + 1  # You can have used 0, 1, ..., all morphemes.
    n_cols_trellis = n_chars + 1  # Column i shows the best path to get to character i (starting at 0).
                                  # You need an "end-character" column to traverse the whole string.

    trellis = [  # Note that the trellis is indexed with transposed indices a.o.t. a matrix.
        [ViterbiNode() for _ in range(n_rows_trellis)]
        for _ in range(n_cols_trellis)
    ]
    for n_morphemes_expended in range(n_rows_trellis):
        trellis[0][n_morphemes_expended].best_count = 0  # Better than -1, the default for all the following nodes.

    # Forward pass
    for char_idx in range(n_chars):  # The last column isn't solved, but only stored in.
        for n_morphemes_expended in range(n_rows_trellis):
            # You now know which search node you are at. You will
            # now try to offer yourself to all reachable nodes.
            current_node = trellis[char_idx][n_morphemes_expended]

            if n_morphemes_expended < n_morphemes:
                # Reachable set 1: anything an available prefix allows.
                for prefix in morpheme_prefices[n_morphemes_expended]:
                    if lemma_normed[char_idx:].startswith(prefix):
                        # You offer yourself to the node with 1 more
                        # morphemes expended and one prefix ahead.
                        amount_covered = len(prefix)
                        score_after_step = current_node.best_count + amount_covered

                        new_char_idx = char_idx + amount_covered
                        new_n_morphemes = n_morphemes_expended + 1

                        new_node = trellis[new_char_idx][new_n_morphemes]
                        if new_node.best_count < score_after_step:
                            new_node.best_count = score_after_step
                            new_node.backpointer = (char_idx, n_morphemes_expended)

                # Reachable set 2: skipping any amount of characters.
                for new_char_idx in range(char_idx + 1, n_cols_trellis):
                    # Don't allow dropping a morpheme. The reason is
                    # that a node already attempts to do that itself
                    # by moving vertically in the table.
                    new_node = trellis[new_char_idx][n_morphemes_expended]
                    if new_node.best_count < current_node.best_count:
                        new_node.best_count = current_node.best_count
                        new_node.backpointer = (char_idx, n_morphemes_expended)
            else:  # You can only skip. It is pointless to skip in many steps, so go right to the end.
                new_node = trellis[-1][-1]
                if new_node.best_count < current_node.best_count:
```

```
                    new_node.best_count = current_node.best_count
                    new_node.backpointer = (char_idx, n_morphemes_expended)



    # Backward pass
    # - Find best node in the last column by maxing on a double key:
    #   in case of a tie, the one with the most morphemes expended wins.
    col_idx = n_cols_trellis - 1
    row_idx = max(range(n_rows_trellis), key=lambda row: (trellis[col_idx][row].best_count, row))
    node = trellis[col_idx][row_idx]

    # - Build string
    morph_split = ""
    alignment = []
    while node.backpointer is not None:
        new_col_idx, new_row_idx = node.backpointer

        is_start_of_morpheme = new_row_idx != row_idx and new_col_idx != col_idx
        # => You consumed a morpheme, and more than 0 characters of it.
        morph_split = " "*(is_start_of_morpheme and new_col_idx != 0) +\
                          lemma[new_col_idx:col_idx] + morph_split
        # => If you stayed on the same row, the added substring was caused by a skip, not a prefix.
        if is_start_of_morpheme:
            alignment.append(new_row_idx)
        elif new_col_idx == 0:  # Skip to the start. Special case where lemma doesn't start with any morpheme.
            alignment.append(None)  # Dummy alignment; probably assign this to the first morpheme.

        col_idx, row_idx = new_col_idx, new_row_idx
        node = trellis[col_idx][row_idx]
    alignment.reverse()

    return morph_split, alignment
```

C Datasets

Deze appendix geeft een korte beschrijving van de datasets die gebruikt zijn tijdens de experimenten van Hoofdstuk 4.

C.1 OSCAR

C.1.1 Wat is het?

OSCAR is een corpus van teksten op het web die door de webscraper van CommonCrawl automatisch verzameld zijn. De teksten zijn gesorteerd per taal door Suárez e.a. (2019), en ik gebruik de Nederlandse split (`unshuffled-deduplicated-nl`) van de 2019-editie.

C.1.2 Hoe verwerk ik het?

Aangezien ik geen modellen – taalmodellen of tokenisers – overheen volledige zinnen train en alleen individuele woorden gebruik, splits ik eerst alle zinnen van OSCAR op spaties, om dan te tellen hoeveel keer elke string die zo verschijnt, voorkomt. Het resultaat schrijf ik naar een bestand met op elke lijn een string, een spatie, en dat aantal.

Vervolgens loop ik door dat bestand en verwijder ik alle woorden die voldoen aan een van de volgende voorwaarden:

- Langer dan 60 karakters. Het langste geldige Nederlandse woord dat ik manueel heb gevonden met de zoekfunctie van Visual Studio Code was 44 karakters lang ("*toezichthoudersaansprakelijkheidsverzekering*"). De regex `[A-z]45[A-z]* [0-9]` heeft nog steeds meer matches dan VS Code kan tellen ($\geq$20 000), maar de meeste lijken afval. 60 is een veiligheidsmarge.
- Bevat disjuncte getallen, matchend met de regex `[A-z]+[0-9]+[A-z]+[0-9]+[A-z0-9]*`. Het gaat dan vaak over online-gebruikersnamen. Een uitzondering zijn chemische formules (bv. H_2CO_3), maar die hebben net als getallen geen linguïstische morfologie.
- Bevat een underscore.
- Matcht een van de regex'en voor exotische alfabetten in Tabel C.1 (afgeleid van o.a. hun Wikipediapagina's).

Taal	Reguliere expressie
Chinees	`[\u4e00-\u9fff]`
Japans	`[\u4E00-\u9FBF]|[\u3040-\u309F]|[\u30A0-\u30FF]`
Koreaans	`[\u1100-\u11FF]|[\u3130-\u318F]|[\uA960-\uA97F]|[\uD7B0-\uD7FF]`
Hebreeuws	`[\u0590-\u05FF]`
Arabisch	`[\u0600-\u06FF]`
Grieks	`[\u0370-\u03FF]`
Indisch	`[\u0900-\u097F]|[\u0980-\u09FF]|[\u0A80-\u0AFF]|[\u0D00-\u0D7F]`
Thais	`[\u0E00-\u0E7F]`
Sri-Lankaans	`[\u0D80-\u0DFF]`
Laotiaans	`[\u0E80-\u0EFF]`
Privé-regio van Unicode	`[\uE000-\uF8FF]`
Pictogrammen en emoji	`[\u25a0-\u27bf]|\ud83c[\ud000-\udfff]|\ud83d[\ud000-\udfff]|\ud83e[\ud000-\udfff]`

Tabel C.1 – Regex'en voor de Unicode-intervallen van exotische alfabetten.

Het resultaat zijn 21 885 652 unieke strings met hun frequentie. De grootte van OSCAR impliceert tegelijk dat veel van die strings geen echte woorden zijn, maar van de strings die dat wel zijn, is de frequentie een stabiele schatter. Vandaar is OSCAR bv. bruikbaar om frequenties op te zoeken voor manueel samengestelde woordenlijsten zoals die hieronder.

C.2 e-Lex

C.2.1 Wat is het?

e-Lex is de laatste nieuwe versie van het "lexicon voor taal- en spraaktechnologie" (TST-lexicon) van de Taalunie (2014). Het brengt data uit verschillende bronnen samen over zowel de morfologie als de uitspraak van Nederlandse woorden.

e-Lex bestaat in `.txt`- en in `.xml`-formaat. De `.txt` heeft een eerder `.csv`-achtige layout waarbij voor elk woord in het lexicon alle 19 datakolommen zijn opgelijst en aldus grotendeels leeg zijn. De `.xml` werkt veel handiger aangezien het een hiërarchische structuur heeft: het hoogste niveau is een lijst van lemmata, en de kinderen van elk lemma zijn o.a. zijn morfologische analyse uit CELEX en alle verbogen woordvormen die tot hetzelfde lexeem behoren.

De woordvormen bevatten ook woordfrequenties, maar uit een kleiner corpus dan OSCAR (het woord *de* heeft bv. frequentie 291 059 502 in OSCAR maar slechts 248 171 in e-Lex). Tabel C.2 geeft een overzicht van hoeveel (unieke) lemmata en (unieke) woordvormen e-Lex bevat.

	Aantal
Woordvormen	615 120
Unieke woordvormen	403 580
Lemmata	216 973
Unieke lemmata	203 466

Tabel C.2 – Woordstatistieken in e-Lex.

C.2.2 Hoe verwerk ik het?

Van de 19 kolommen in e-Lex heb ik er slechts 2 nodig: het lemma en de morfologische analyse van het lemma. Van de 203 466 unieke lemmata zijn er 96 431 (47%) voorzien van zo'n analyse. Tabel C.3 toont dat er in totaal 98 035 morfologische analyses beschikbaar zijn; daarvan zijn er 974 volledige duplicaten, en nog 521 zijn hetzelfde modulo hun structuur (d.w.z.: ze hebben dezelfde morfemen en geen verschil in prefix-/suffixgedrag, maar hebben redundantie in hun PoS-labels, bv. het substantief *vierkant* en het adjectief *vierkant*). Er zijn dan nog 96 540 morfologieën over, wat ruim voldoende moet zijn om de woordvorming van de Nederlandse taal mee te vatten. Let wel dat e-Lex geen analyse geeft van verbuigingen en vervoegingen; alleen lemmata.

	Aantal
Unieke morfemen	13 317
Unieke morfen	18 516
Unieke lemmata met morfologie	96 431
Unieke morfeemsequenties	96 523
Unieke structuren	96 540
Unieke morfologieën	97 061
Totale morfologieën	98 035

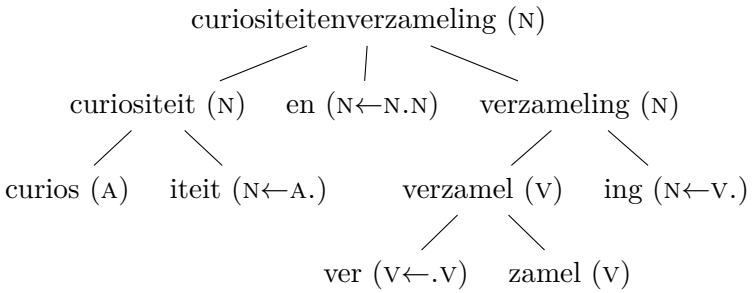
Tabel C.3 – Morfologiestatistieken in e-Lex.

De morfologische analyses hebben nood aan verdere verwerking aangezien ze in CELEX-formaat staan zoals in §4.13 uitgelegd. Het aligneren van de theoretische morfemen met hun effectieve morfen gebeurt door alle bijkomende syntax van de morfologische analyse te negeren en de morfemen als platte lijst te zien, en dan §B.4 toe te passen.

Echter, het CELEX-formaat volgt een contextvrije grammatica die een boom van morfemen voorstelt, niet gewoon een platte lijst. Vandaar zet ik het CELEX-formaat om naar een boomstructuur waarvan de bladeren zowel het morfeem als het respectievelijke (al dan niet lege) morf toegewezen krijgen. Elke knoop in de boom heeft met name een PoS-tag die ook aangeeft of de subboom waarvan die knoop de wortel is een volmaakt lemma is. Neem als voorbeeld de analyse voor *curiositeitenverzameling*:

```
(((curieus)[A],(iteit)[N|A.])[N],(en)[N|N.N],(((ver)[V|.V],(zamel)[V])[V],(ing)[N|V.])[N])[N]
```
(C.1)

De resulterende boom, na het morfeem *curieus* door het morf *curios* te vervangen en op elke knoop de concatenatie van alle bladeren van zijn subboom te tonen, is:



Een PoS-tag van de vorm $C \leftarrow A.B$ betekent "affixmorfeem dat tussen PoS A en PoS B staat en daardoor PoS C produceert". De suffix *-ing* produceert bv. een zelfstandig naamwoord (N) van een werkwoord (V) door erachter te staan.

Dergelijke morfemen kunnen niet als zelfstandig lemma voorkomen – ze behoren tot geen enkel lexeem. Vandaar is het interessant om twee soorten splitsingen te beschouwen:

- *morfemische splitsing*: splits een gegeven woord op in morfen.
  Hierboven: *curios|iteit|en|ver|zamel|ing*.
- *lexemische splitsing*: splits een gegeven woord op in de kleinste delen die allemaal hun eigen lexeem hebben. Hierboven: *curiositeiten|verzameling*.

Vanwege de variabiliteit van interfices kies ik ervoor om ze ook in de lexemische splitsing los te koppelen: *curiositeit|en|verzameling*. Een gelijkaardig idee komt in §2.4.1 aan bod bij het bespreken van het systeem van Fritzinger en Fraser (2010).

Een algoritme dat lexemisch splitst, bekijkt de boom van boven naar onder; indien minstens één van de kinderen van een morf een pre- of suffix is, wordt de hele subboom verwijderd. In de bovenstaande boom wordt de subboom onder *curiositeit* verwijderd door *-iteit* en die onder *verzameling* door *-ing*.

C.3 NT2Lex

C.3.1 Wat is het?

NT2Lex van Tack e.a. (2018) bestaat uit 14 724 Nederlandse lemmata met frequentiestatistieken uit teksten van verschillende moeilijkheidsgraden. Van die lemmata zijn er 5967 (40%) die geen morfologie hebben in e-Lex.

C.3.2 Hoe verwerk ik het?

Ik extraheer de lemmata en gooi de rest van de informatie weg. Gezien er geen morfologische annotatie in NT2Lex zit, gebruik ik die lemmata zonder verdere verwerking als invoer voor de typo-experimenten in §4.15.

C.4 Vulgaire woorden

In §4.8 onderzoek ik het effect van de aanwezigheid van teksten van pornografische websites in het Nederlandse deel van OSCAR. Daartoe is een lijst van vulgaire – en specifiek pornografische – Nederlandse woorden nodig. Er is bij mijn weten geen centrale autoriteit die een dergelijke lijst bijhoudt, dus stel ik er zelf een op.

Scheldwoorden Aangezien lijsten van scheldwoorden vaak toepassing vinden in censuur op het web, zijn de volgende bronnen beschikbaar als `node.js`-pakket:

- `naughty-words[nl]` van Shutterstock: manueel samengestelde lijst van scheldwoorden uit Nederland.
- `washyourmouthoutwithsoap[nl]`: automatische vertalingen van de Engelse `badwords` van Google.
- `profane-words`: compilatie van meerdere Engelse scheldwoordlijsten.

De lingua franca op het web is het Engels, dus zijn er mogelijks enkele Engelse websites in het Nederlandse deel van OSCAR geslopen. Vandaar dat ik vertrek van de concatenatie van de Engelse en Nederlandse lijsten hierboven.

Onder vulgariteiten vallen meer dan alleen pornografische woorden (bv. *godverdomme*, *tyfus*, *hufter*, *pis* ...), maar die concentreren zich niet op specifieke websites en zijn

dus niet nuttig om domeinspecificatie mee te herkennen. Ik filter dergelijke algemene scheldwoorden dus manueel uit de lijst.

Pornografie We hebben nu scheldwoorden weggefilterd die niet pornografisch zijn. Anderzijds zijn er pornografische woorden die geen scheldwoorden zijn (*erotisch*, *MILF*, *pornoster* ...), waaronder een groep woorden die zowel in normale als in pornografische contexten bruikbaar zijn maar *gegeven* pornografische websites een grote kans hebben om te verschijnen net als die andere, bv. *brunette*, *latina*, *ebony*, *interraciaal*, *POV*, *amateur* enzovoort.

Vandaar vul ik de lijst aan met nog twee informatiebronnen. Enerzijds laat ik een webscraper alle videocategorieën van de homepagina van `https://www.porn.com/`[72] ophalen, wat gemakkelijk gaat aangezien ze in alfabetisch gegroepeerde HTML-lijsten staan. Anderzijds laat ik een scraper alle namen van pornowebsites verzamelen die op `https://toppornsites.com/`[73] ook in HTML-lijsten gestructureerd zijn.

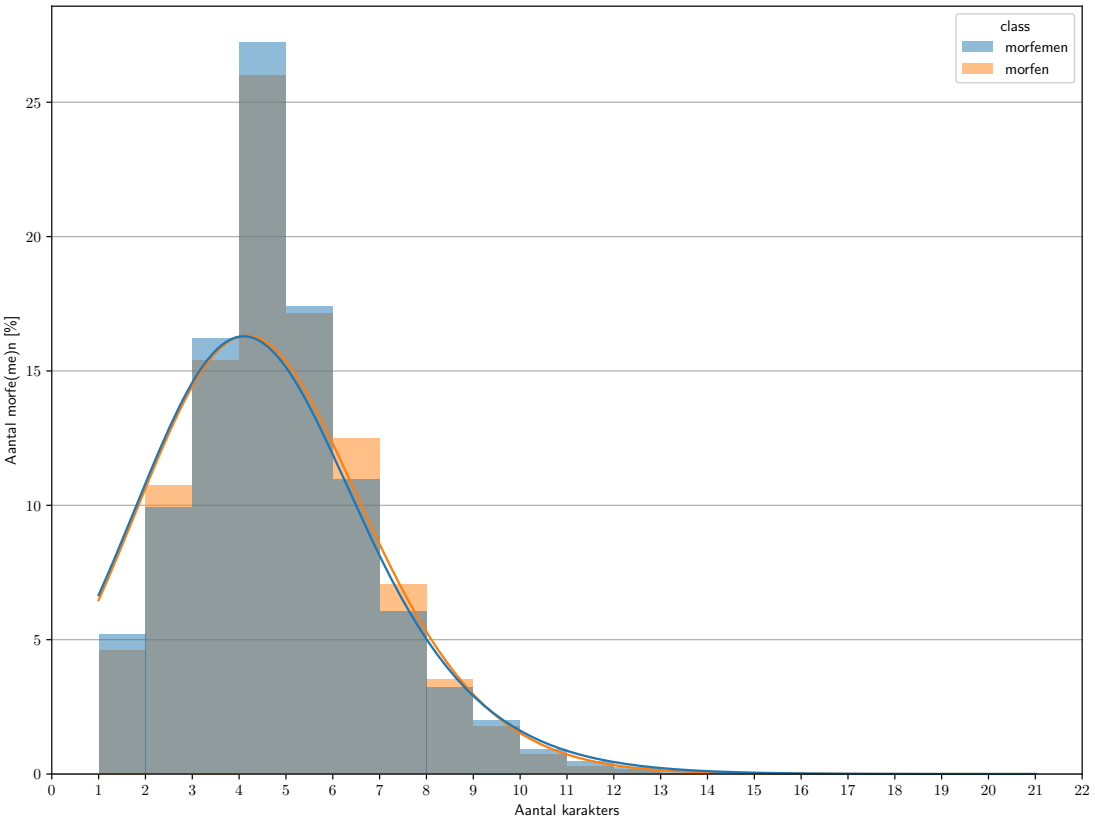
Alles samen is de lijst van vulgaire woorden zo'n 1700 termen lang.

[72] Gekozen vanwege de generieke naam. Bezoek op eigen risico.
[73] Gekozen vanwege de lengte. Bezoek op eigen risico.

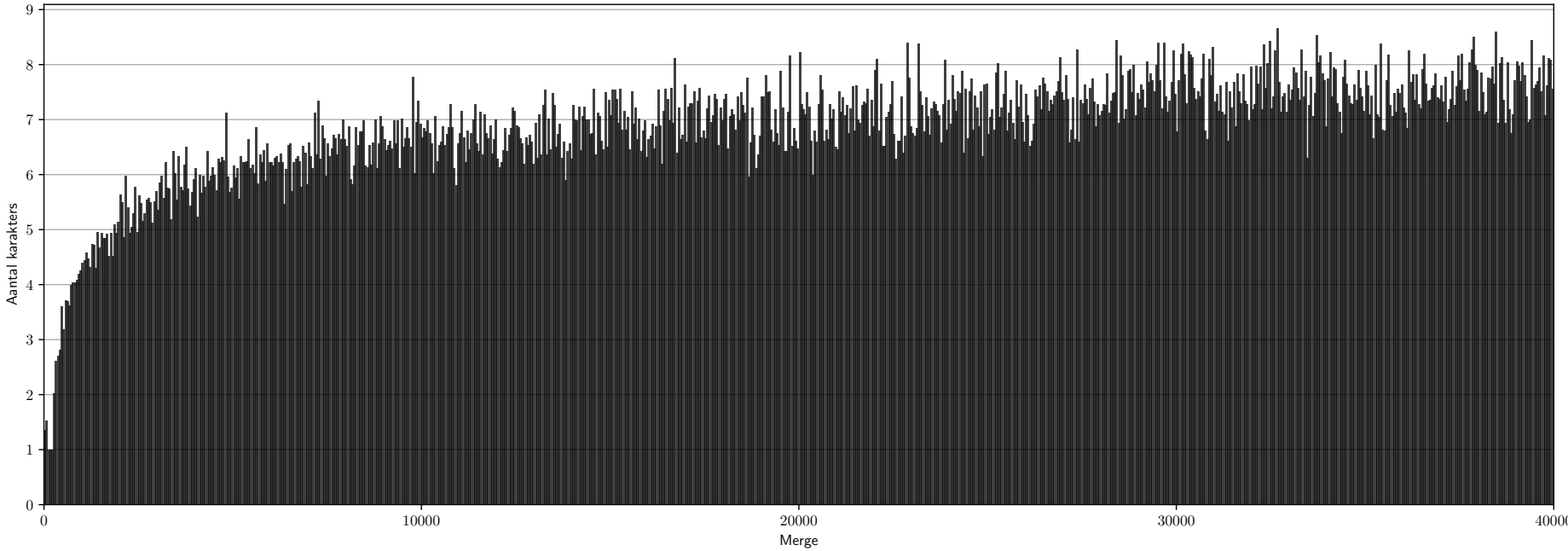
D Extra figuren

Deze appendix bevat bijkomende figuren waarnaar verwezen wordt in de tekst.

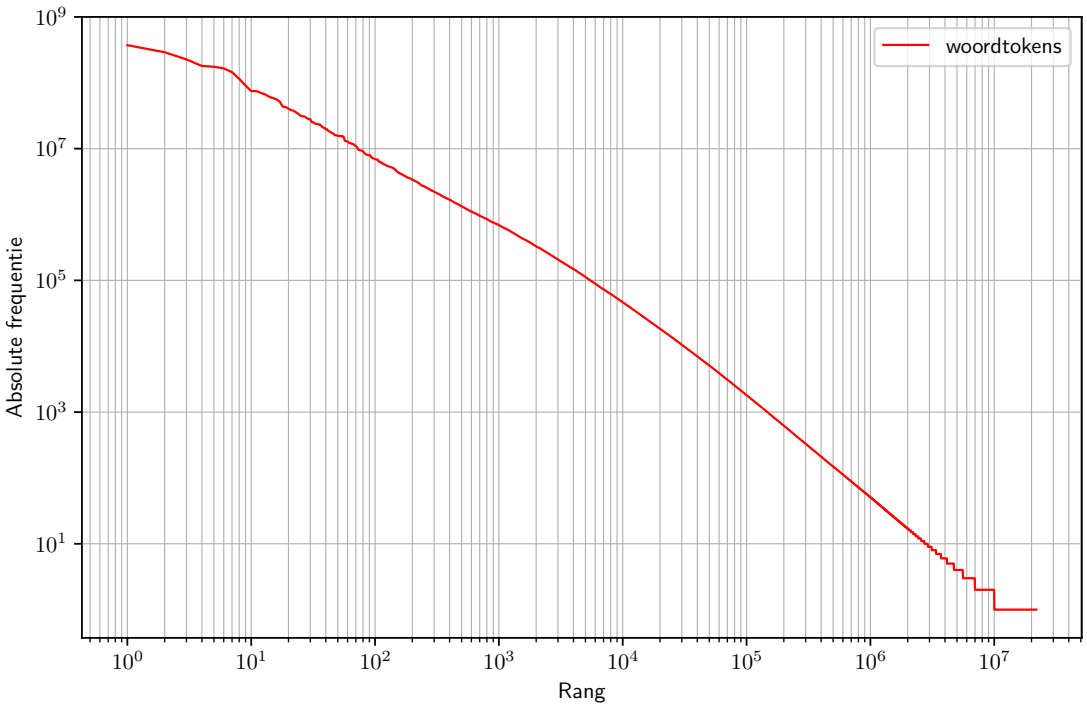
D.1 Stellingen over taal



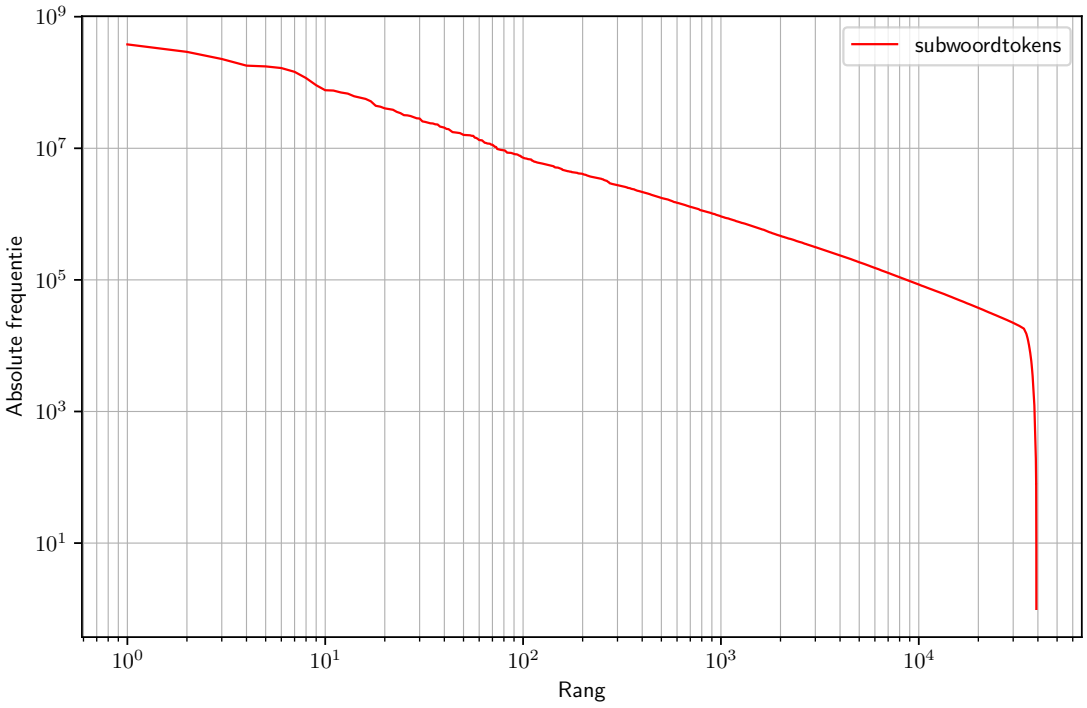
Figuur D.1 – "Morfemen zijn kleine strings groter dan een karakter."



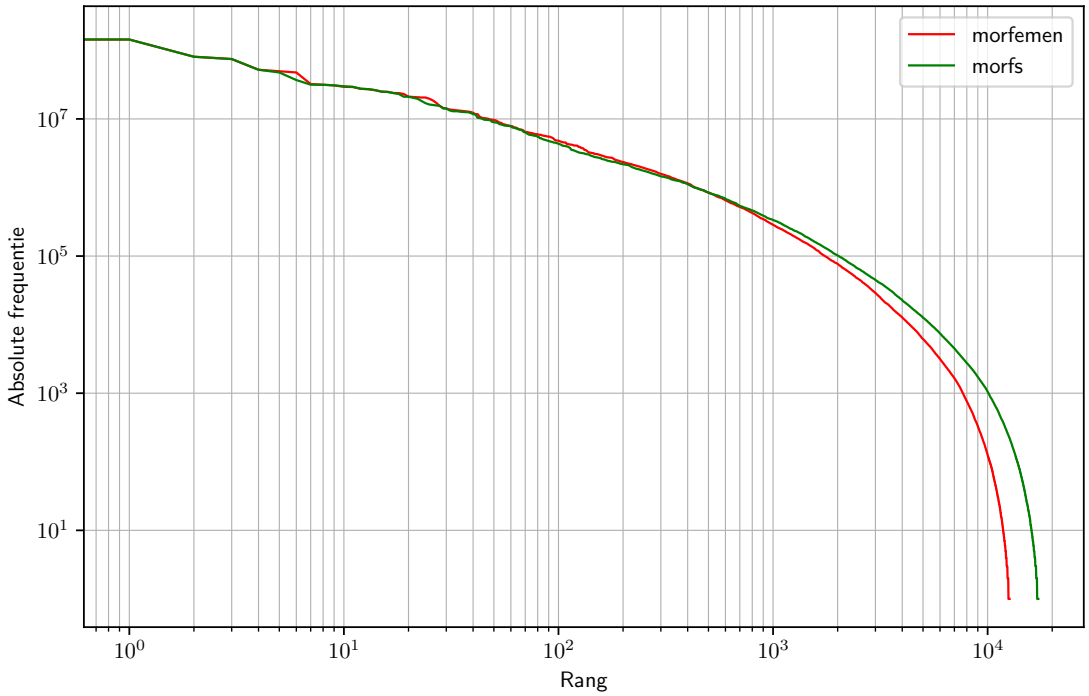
Figuur D.2 – "BPE-types worden groter naarmate het vocabularium (aantal merges) stijgt."



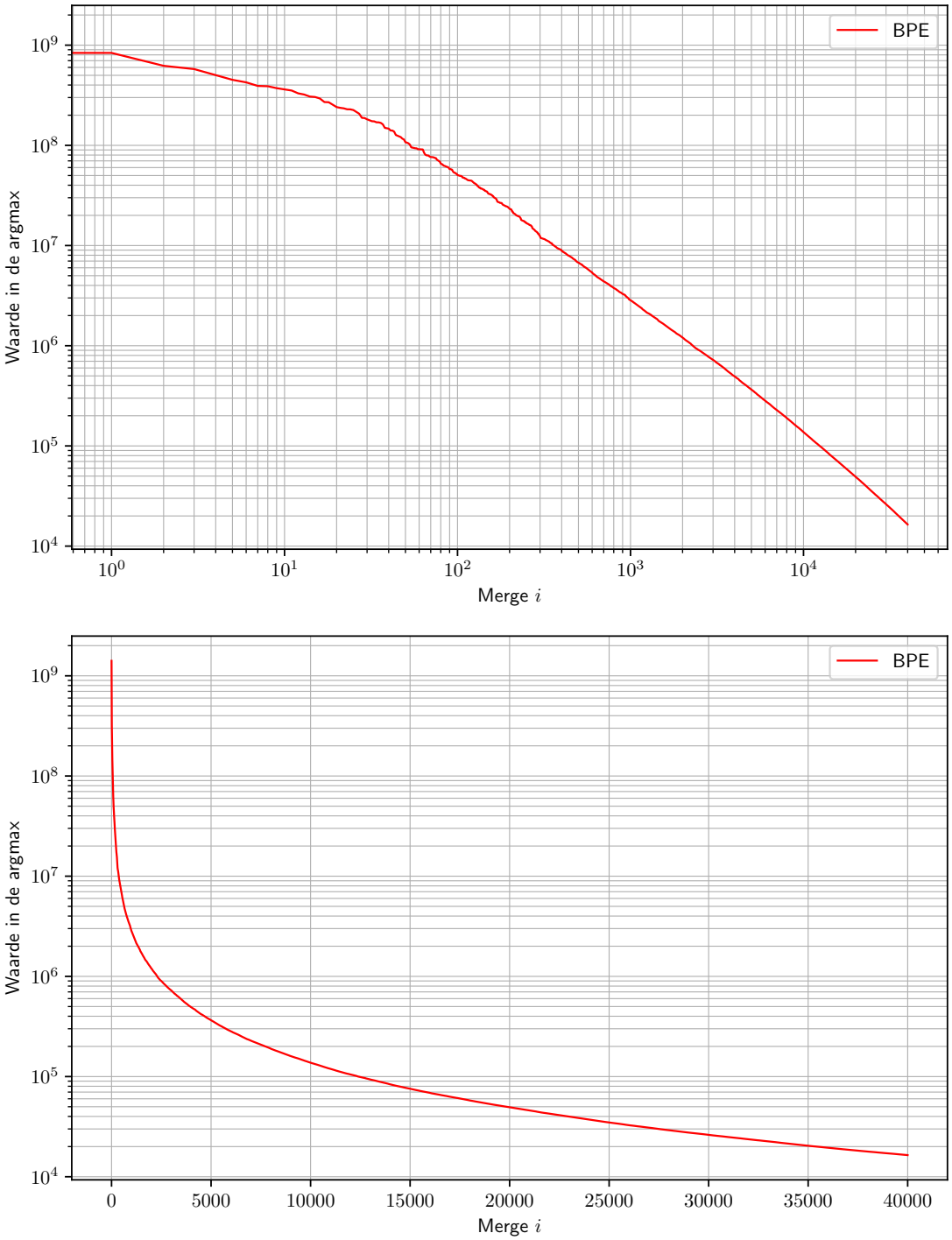
Figuur D.3 – "Woorden zijn zipfiaans verdeeld." Gesorteerde woordfrequenties in OSCAR.



Figuur D.4 – "Subwoordtokens zijn zipfiaans verdeeld." Gesorteerde BPE_{nl}-subwoordfrequenties in OSCAR.

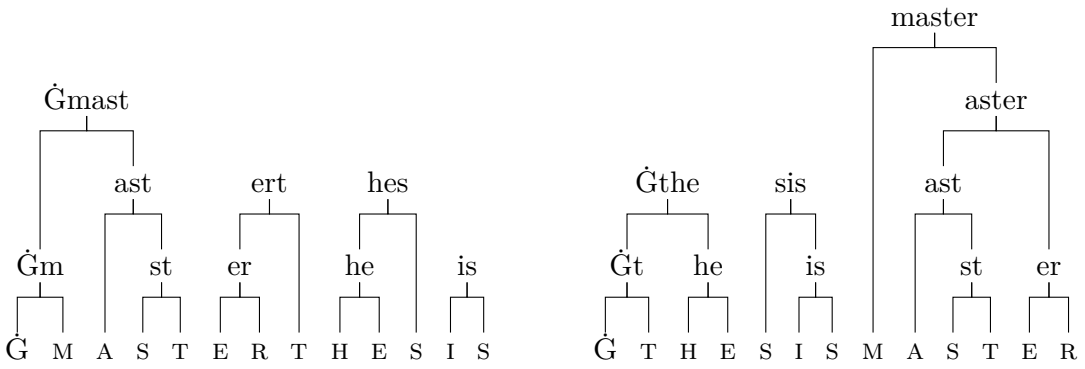


Figuur D.5 – "Morfemen en morfen zijn zipfiaans verdeeld." Gesorteerde morf- en morfeemfrequenties in e-Lex-lemmata gewogen met hun frequentie in OSCAR.

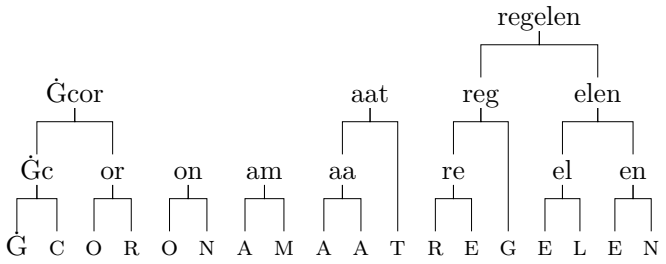


Figuur D.6 – "De waarden van de BPE-argmax (lijn 7) zijn zipfiaans verdeeld."

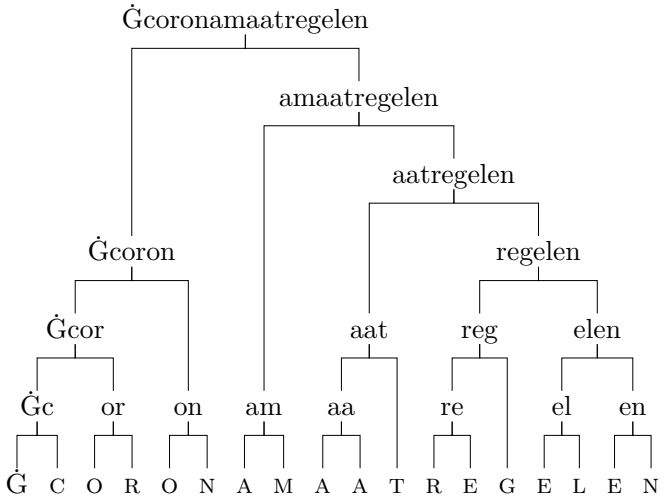
D.2 Tokenisatievoorbeelden



Figuur D.7 – BPE-tokenisatie van *masterthesis* en *thesismaster*.

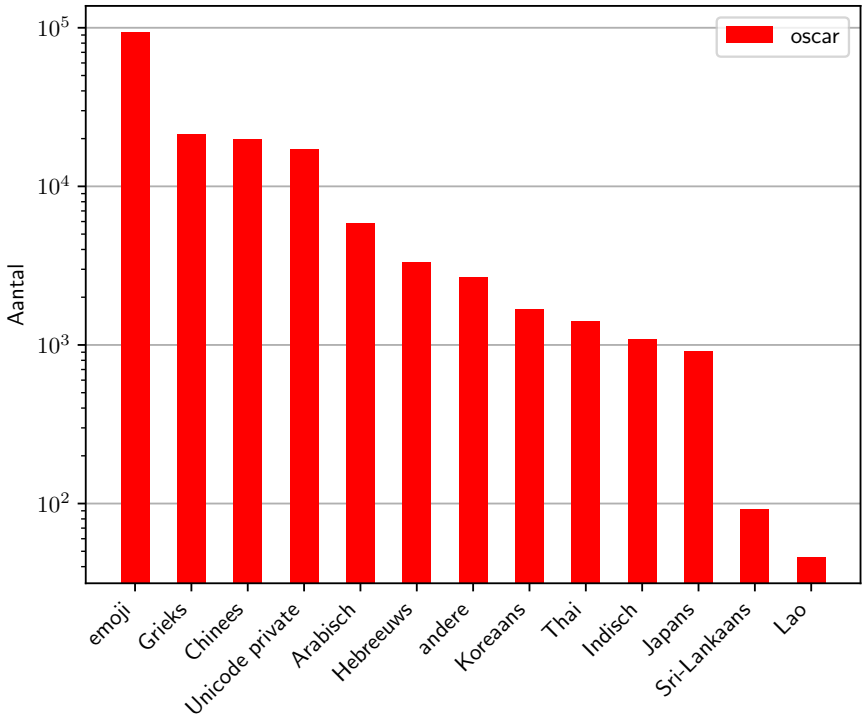


Figuur D.8 – Tokenisatie van *coronamaatregelen* door RobBERT-2020.



Figuur D.9 – Tokenisatie van *coronamaatregelen* door RobBERT-2022.

D.3 Unicode



Figuur D.10 – Aantal fragmenten in OSCAR-nl met een exotisch teken, gegroepeerd op de categorie van het eerste gevonden teken.

Id	Type	Id	Type	Id	Type	Id	Type
16847	Ġ(âĢ¦)	17028	âĢ¯	17179	.âĢĻâĢĻ	17195	çķĦ
17364	ĠâĢ	17657	âĢĽ)	17786	ĐµÑĤ	17910	.âĢ¦
18007	Đ°ÑĤ	18059	Ġ×	18100	ĠÏ	18120	ĠðŁĺ
18146	ÑģĐº	18205	ÜĽ	18268	ä¹	18332	ãģ®
18667	ĠÅ	18809	Ãī	18829	ĠðŁ	18893	ĠÃĺ
18986	Đ½Ñĭ	19062	ãĢĤ	19119	ÑĪ	19309	à¹
19347	ÃŁ	19550	çĶ	19732	Ñ	19779	ä¸Ń
19939	Üĺ	20070	ï¼	20113	Đ¸ÑĤ	20179	åĽ
20474	ĠâĢ¦.	20483	æ	20615	Ġ...âĢĽ	20751	ÑĪ
20890	Ñİ	20924	åIJ	21013	ĠĐ¿ÑĢ	21227	ĠØ§ÙĦ
21296	â	21299	ĠÆĴ	21319	ë	21337	Ùĩ
21614	Ġâ	21726	åĪ	21782	ï	21862	ĠâĢĺâĢĻ
22029	ĠÂij	22120	à¥	22501	[âĢ¦]	22735	âĢĻ?
22911	ĠâĢĭ	23093	ĠÑĢ	23098	âĢ¨	23147	Ġà¤
23211	ĠÃī	23266	Äį	23304	ÃŃa	23464	Đ°ÑĢ
23658	ĠÑĤ	23667	Đ¸Ñı	23714	Ġç	23749	åľ
23757	âĢĻ).	23847	è®	23856	ãģĦ	23875	ĠÃĥ

Tabel D.1 – Enkele byte-gebaseerde types in RobBERT-2022 met onbekende herkomst.

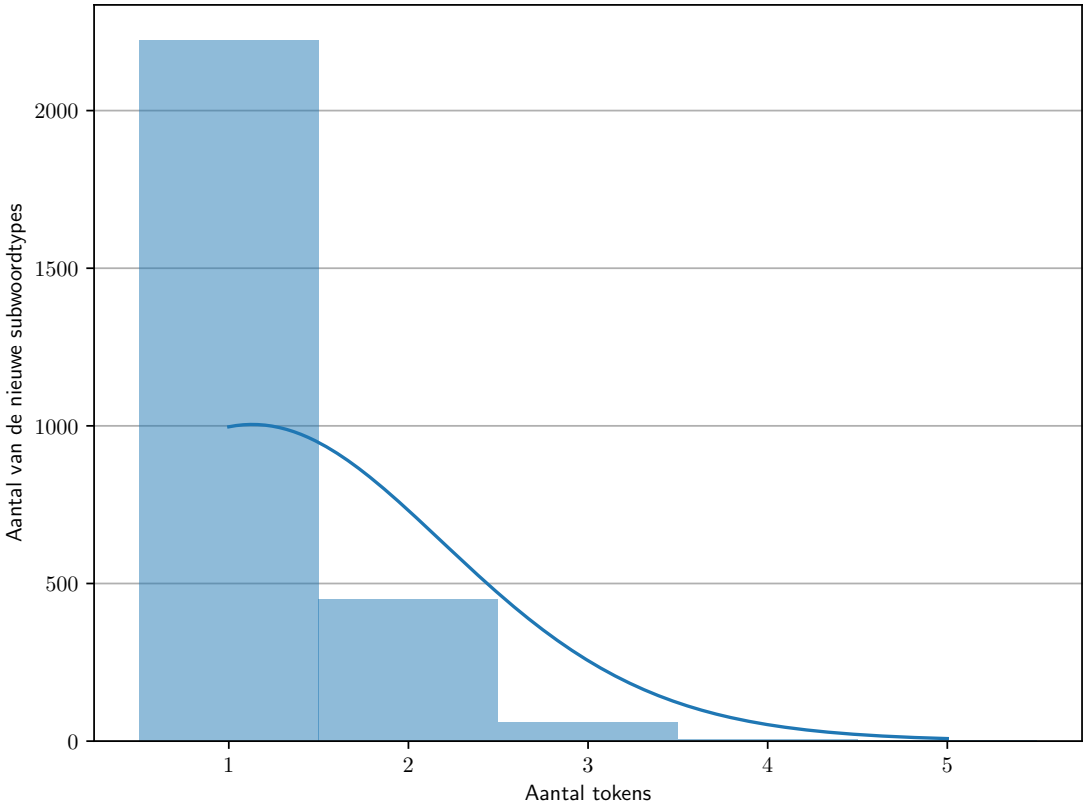
D.4 Dataonbalans

Tabel D.2 – Waarschuwing: verbrand uw communiezieltje niet aan deze tabel. Types t in het BPE_{nl}-vocabularium (zwarte tekst in de eerste kolom van de twee panelen) met overlap in de vulgairewoordenlijst (VW) van minstens 5 karakters of het volledige vulgaire woord, geordend op aantal voorkomens f_s als substring in e-Lex, samen met het meest frequente woord w in OSCAR waarvoor $t \in w$. Manueel gefilterd.

in VW	f_s e-Lex	in OSCAR, w	$C(w)$	in VW	f_s e-Lex	in OSCAR, w	$C(w)$
ass	1483	massage	1 159 837	Ġcum	158	cum	106 232
Ġborsten	140	borsten	278 103	borsten	140	borsten	278 103
Ġmassage	125	massage	1 159 837	massage	125	massage	1 159 837
Ġanaal	116	anaal	158 062	anaal	116	anaal	158 062
Ġpron	115	pron	22 285	Ġzaad	112	zaad	96 253
zaad	112	zaad	96 253	amateur	106	amateur	242 959
Ġcialis	104	cialis	29 829	Ġpik	99	pik	207 071
Ġspuiten	95	spuiten	110 126	spuiten	95	spuiten	110 126
Ġseks	90	seks	1 227 174	seks	90	seks	1 227 174
Ġhomo	77	homo	167 554	Ġreet	75	reet	34 287
lingerie	69	lingerie	87 249	Ġrukken	65	rukken	20 910
Ġkut	64	kut	302 125	Ġsex	63	sex	2 284 225
sex	63	sex	2 284 225	Ġstripper	55	strip	67 299
stripper	55	strip	67 299	Ġlul	54	lul	314 646
amateur	54	amateur	242 959	lul	54	lul	314 646
Ġhoes	52	hoes	83 627	Ġhoer	48	hoeren	89 318
Ġamateur	48	amateur	242 959	Ġtrio	44	trio	113 322
erotisch	44	erotische	449 352	massage	44	massage	1 159 837
ejaculatie	43	ejaculatie	16 663	Ġporn	42	porno	1 231 265
Ġdomination	42	dominante	40 466	porn	42	porno	1 231 265
Ġsolo	41	solo	94 895	Ġporno	41	porno	1 231 265
porno	41	porno	1 231 265	Ġwip	37	wip	9298
Ġkont	37	kont	321 406	kont	37	kont	321 406
Ġneger	30	negeren	53 022	Ġnaakt	30	naakt	317 813
naakt	30	naakt	317 813	Ġanus	29	anus	42 476
anus	29	anus	42 476	Ġhoeren	28	hoeren	89 318
Ġmassage	27	massage	1 159 837	Ġballen	27	ballen	116 751
massage	27	massage	1 159 837	ballen	27	ballen	116 751
Ġprostituee	26	prostitutie	37 319	Ġpijpen	26	pijpen	158 853
Ġaftrekken	26	aftrek	68 391	pijpen	26	pijpen	158 853
aftrekken	26	aftrek	68 391	Ġnaaien	23	naaien	38 642
Ġspuiten	19	spuiten	110 126	Ġsperma	19	sperma	106 325
spuiten	19	spuiten	110 126	Ġshit	18	shit	26 480
Ġbloot	18	bloot	71 791	Ġzuigen	16	zuigen	94 936
Ġprostituee	16	prostitutie	37 319	Ġlatina	16	latina	77 206
zuigen	16	zuigen	94 936	Ġvagina	15	vagina	82 532
Ġslet	14	slet	78 347	Ġverkrachting	13	verkrachting	36 880
Ġpenis	13	penis	112 552	Ġeikel	12	eikel	35 807
Ġpov	10	pov	30 839	Ġmissionary	10	missionaris	7969
Ġgeil	10	geile	528 090	geil	10	geile	528 090
Ġslikken	9	slikken	79 041	escort	9	escort	348 787
Ġswinger	8	swingers	24 107	Ġorgie	8	orgie	48 323
Ġescort	8	escort	348 787	Ġblow	8	blowjob	21 843
Ġvagina	7	vagina	82 532	Ġpenetrate	7	penetratie	27 778
Ġblote	7	blote	81 836	lingerie	7	lingerie	87 249
Ġverkrachting	6	verkrachting	36 880	Ġteef	6	teef	29 973
Ġneuken	6	neuken	557 639	Ġlingerie	6	lingerie	87 249
Ġclit	6	clitoris	21 514	Ġblacks	6	black	81 912
neuken	6	neuken	557 639	creampie	6	creampie	70 308
Ġcreampie	5	creampie	70 308	cocktease	5	striptease	15 749
Ġpornos	4	pornosterren	40 974	Ġincest	4	incest	17 116
Ġfuck	4	fuck	61 984	ejaculaat	4	ejaculatie	16 663
Ġxx	3	xxx	254 027	Ġtieten	3	tieten	436 187
Ġstandje	3	standjes	32 891	Ġorgasme	3	orgasme	97 351
Ġlatex	3	latex	52 194	Ġhardcore	3	hardcore	99 992
xx	3	xxx	254 027	Ġvoyeur	2	voyeur	18 238
Ġvingeren	2	vingeren	43 410	Ġviagra	2	viagra	62 821
Ġsletten	2	sletten	28 845	Ġsexy	2	sexy	686 441
Ġpanty	2	panty	45 166	Ġmasturbat	2	masturbatie	67 280
Ġgeile	2	geile	528 090	Ġerotisch	2	erotische	449 352
Ġerotiek	2	erotiek	57 588	fucking	2	fucking	26 523

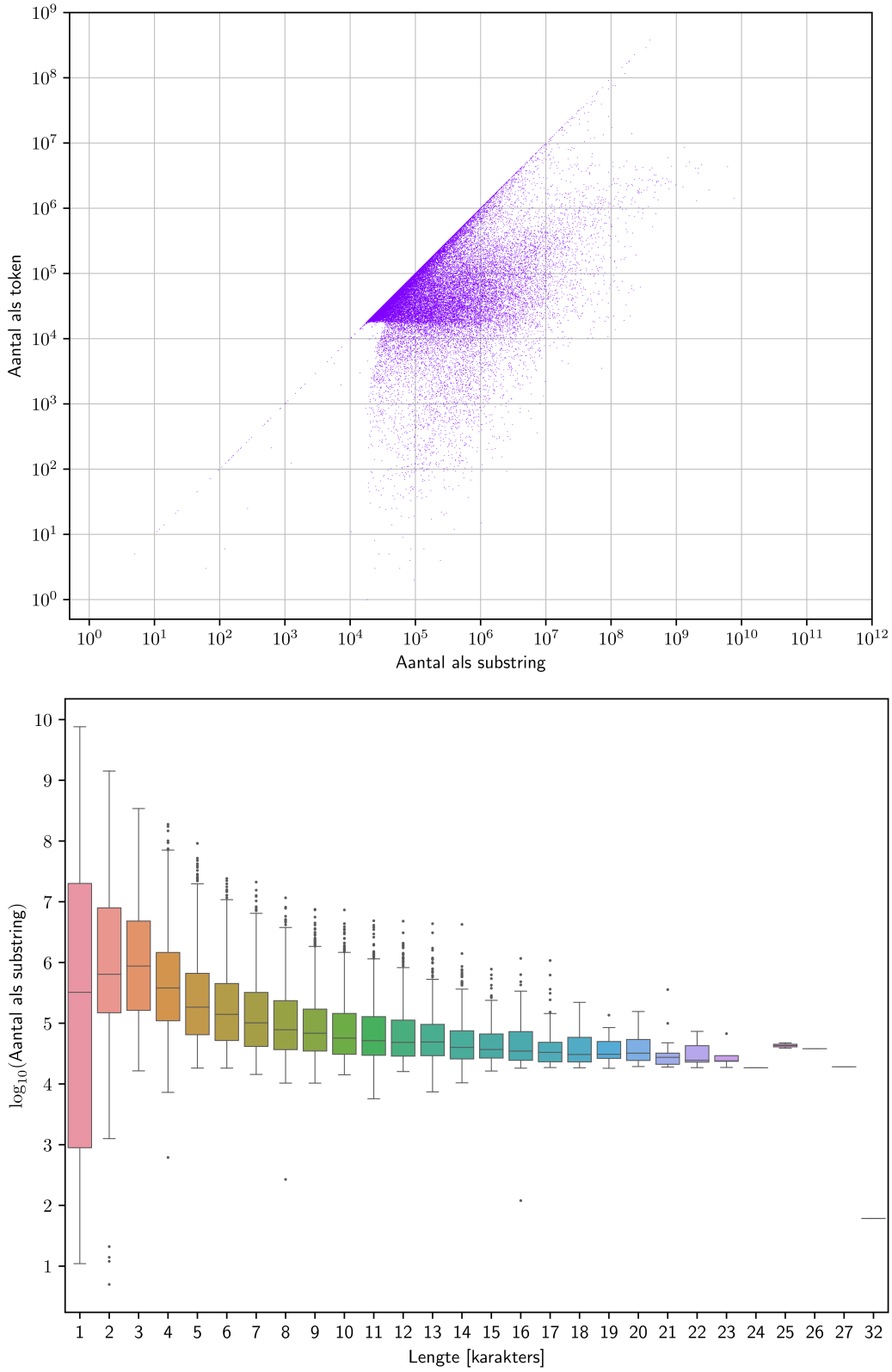
sexy	2	sexy	686 441	voyeur	2	voyeur	18 238
geile	2	geile	528 090	pleasure	2	pleasure	7682
Ġwebcam	1	webcam	189 963	Ġvibrator	1	vibrator	53 376
Ġschaamlippen	1	schaamlippen	20 435	Ġnude	1	nude	20 403
Ġmasturberen	1	masturberen	64 646	Ġklaarkomen	1	klaarkomen	81 003
Ġfucking	1	fucking	26 523	Ġdildo	1	dildo	92 671
Ġclitoris	1	clitoris	21 514	Ġbrunette	1	brunette	52 734
Ġborsten	1	borsten	278 103	Ġbondage	1	bondage	74 499
Ġbeffen	1	beffen	50 048	Ġaftrekken	1	aftrekken	43 401
ejaculatie	1	ejaculatie	16 663	Ġxxx	0	xxx	254 027
Ġxnxx	0	xnxx	26 326	Ġtriootje	0	triootje	27 315
Ġtranny	0	tranny	75 229	Ġswingers	0	swingers	24 107
Ġsquirt	0	squirt	25 139	Ġshemale	0	shemale	144 588
Ġsexcam	0	sexcam	15 888	Ġpussy	0	pussy	36 249
Ġpornstar	0	pornstar	25 036	Ġpijpbeurt	0	pijpbeurt	158 577
Ġneukt	0	neukt	114 833	Ġmilf	0	milf	216 737
Ġmature	0	mature	17 935	Ġmasterb8	0	masterbation	8521
Ġladyboy	0	ladyboy	22 163	Ġkutjes	0	kutjes	50 250
Ġkutje	0	kutje	215 037	Ġkinky	0	kinky	65 081
Ġinterracial	0	interraciale	30 923	Ġhentai	0	hentai	33 291
Ġhandjob	0	handjob	34 510	Ġgirlsgonewild	0	girls	61 047
Ġgangbang	0	gangbang	65 026	Ġfetish	0	fetish	50 585
Ġerotic	0	erotic	21 086	Ġebony	0	ebony	106 433
Ġdoggy	0	doggy	14 789	Ġcuckold	0	cuckold	25 460
Ġcreampie	0	creampie	70 308	Ġblowjob	0	blowjob	21 843
Ġbdsm	0	bdsm	54 626	Ġbbw	0	bbw	123 119
Ġbabes	0	babes	19 526	Ġbabe	0	babes	19 526
xxx	0	xxx	254 027	cuckold	0	cuckold	25 460
shemale	0	shemale	144 588	girlsgonewild	0	girls	61 047

D.5 Domeinadaptatie

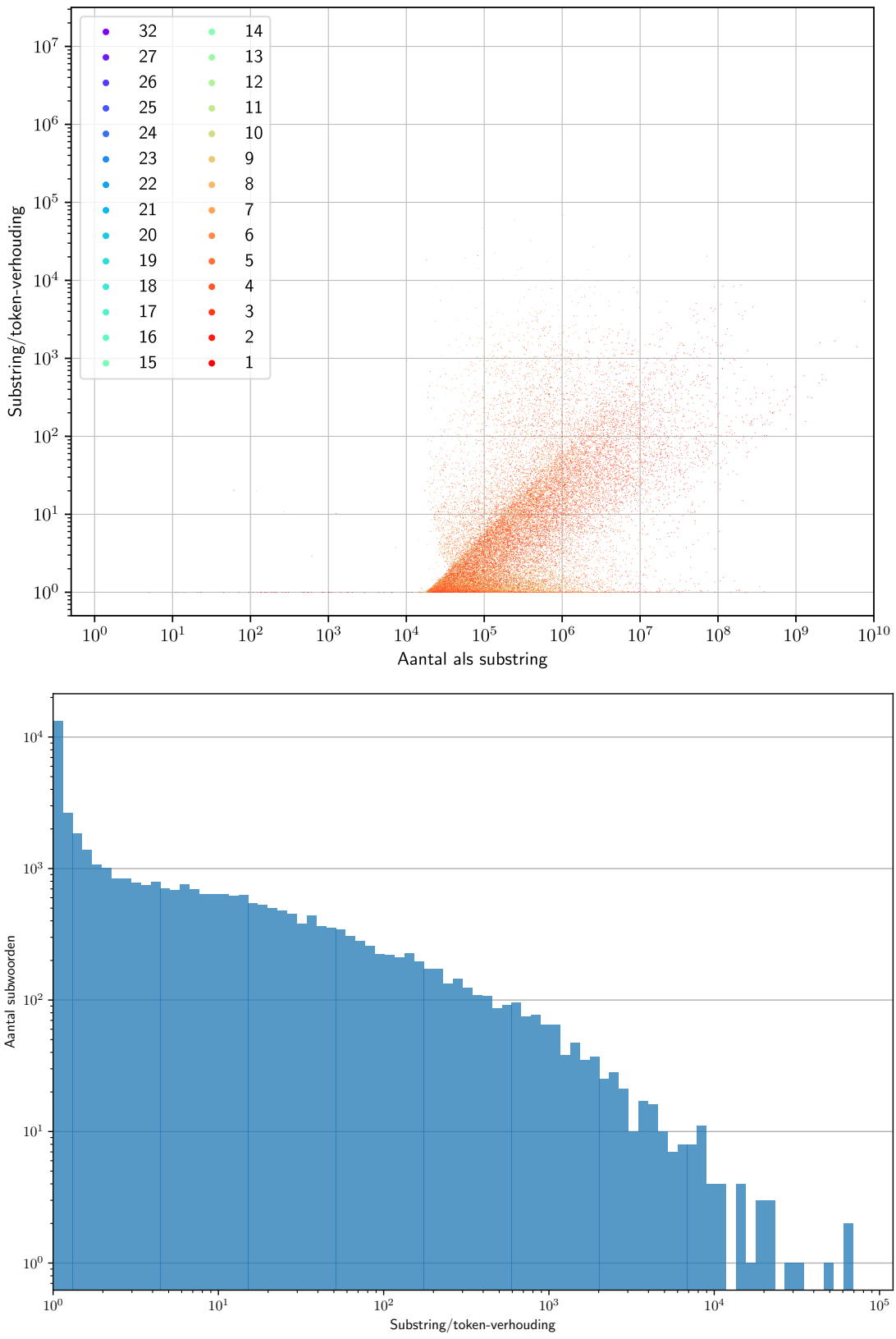


Figuur D.11 – Aantal tokens bij segmentatie van de nieuwe types in RobBERT-2022 met diens tokeniser.

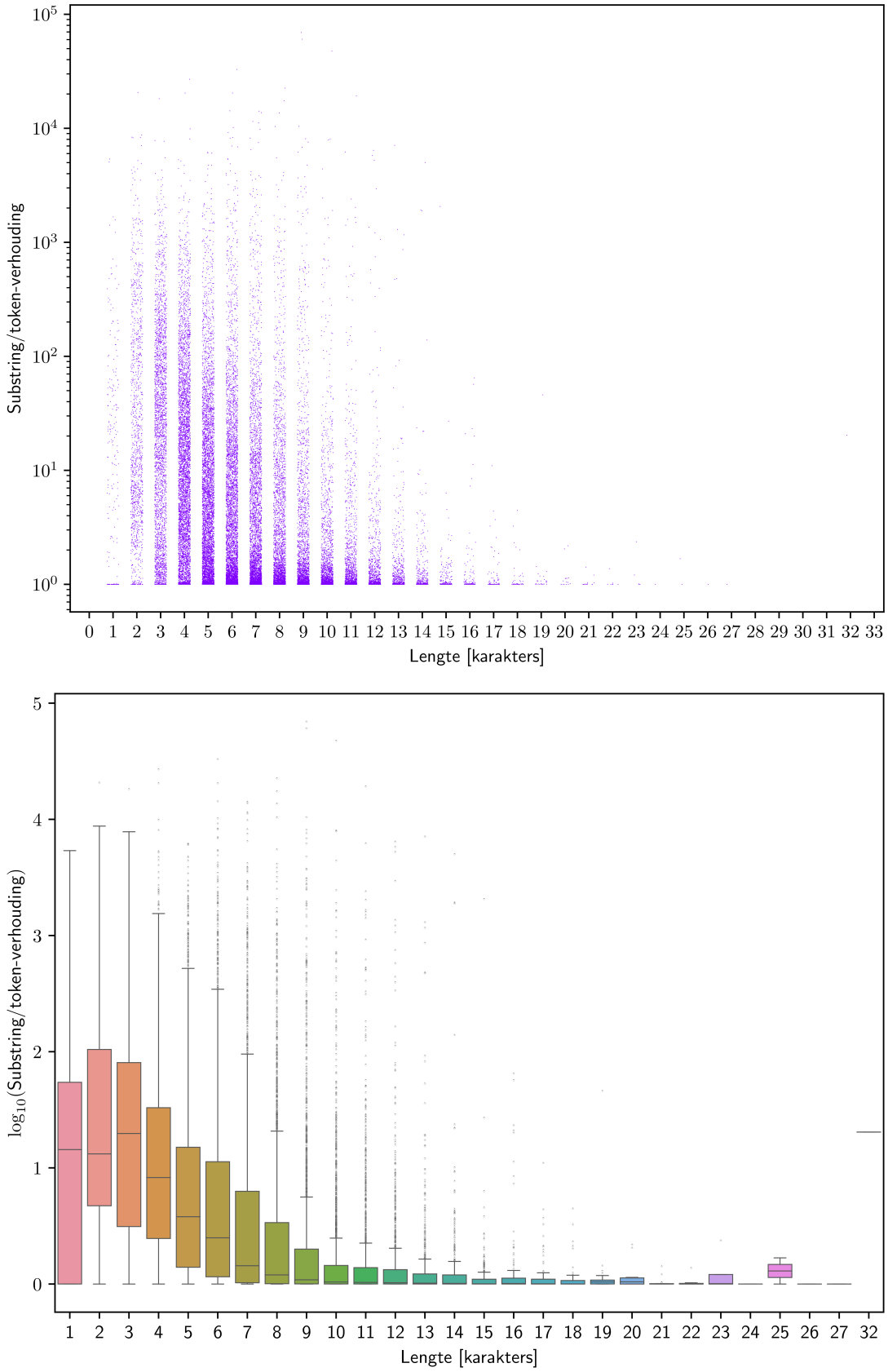
D.6 Datainversie



Figuur D.12 – Vergelijking tussen het aantal voorkomens als token t.o.v. als substring in OSCAR van alle subwoordtypes in het BPE_{nl}-vocabularium.

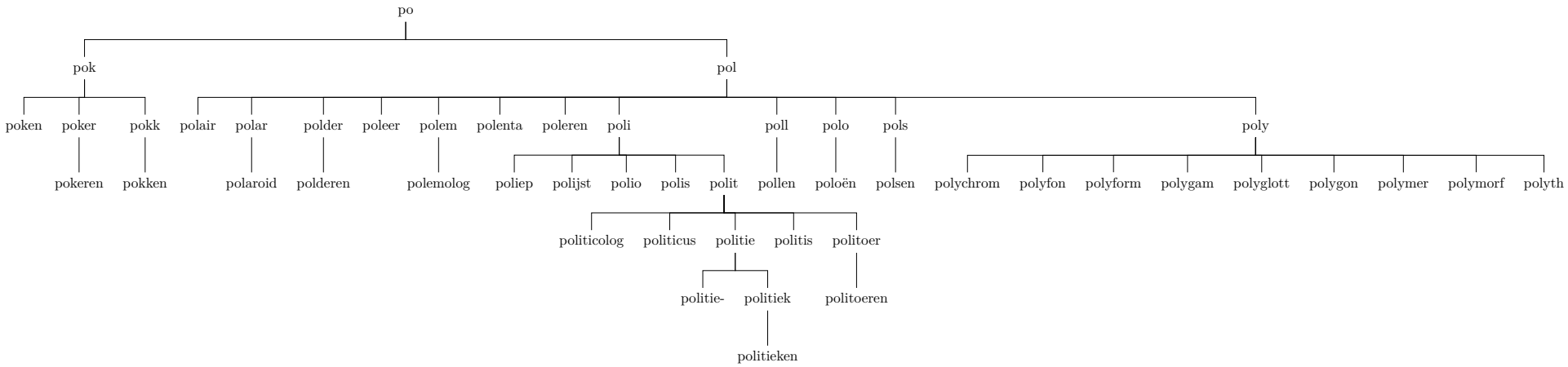


Figuur D.13 – Verdeling van de verhouding tussen substringfrequentie en tokenfrequentie.



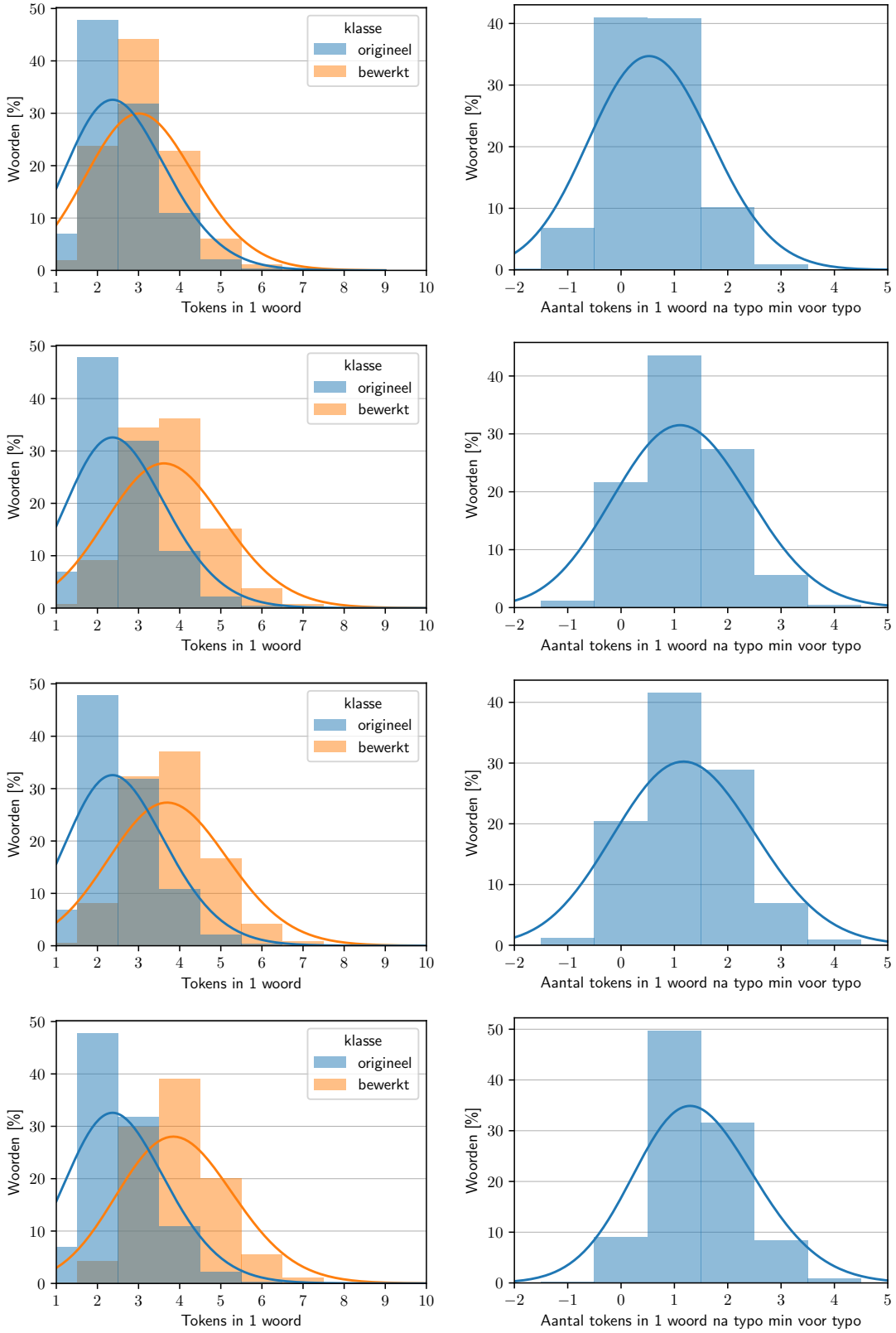
Figuur D.14 – Verdeling van de bovenstaande verhouding ingedeeld per typelengte. Boxplot afgeknot vanaf >1.5 IQR.

D.7 Aliasing

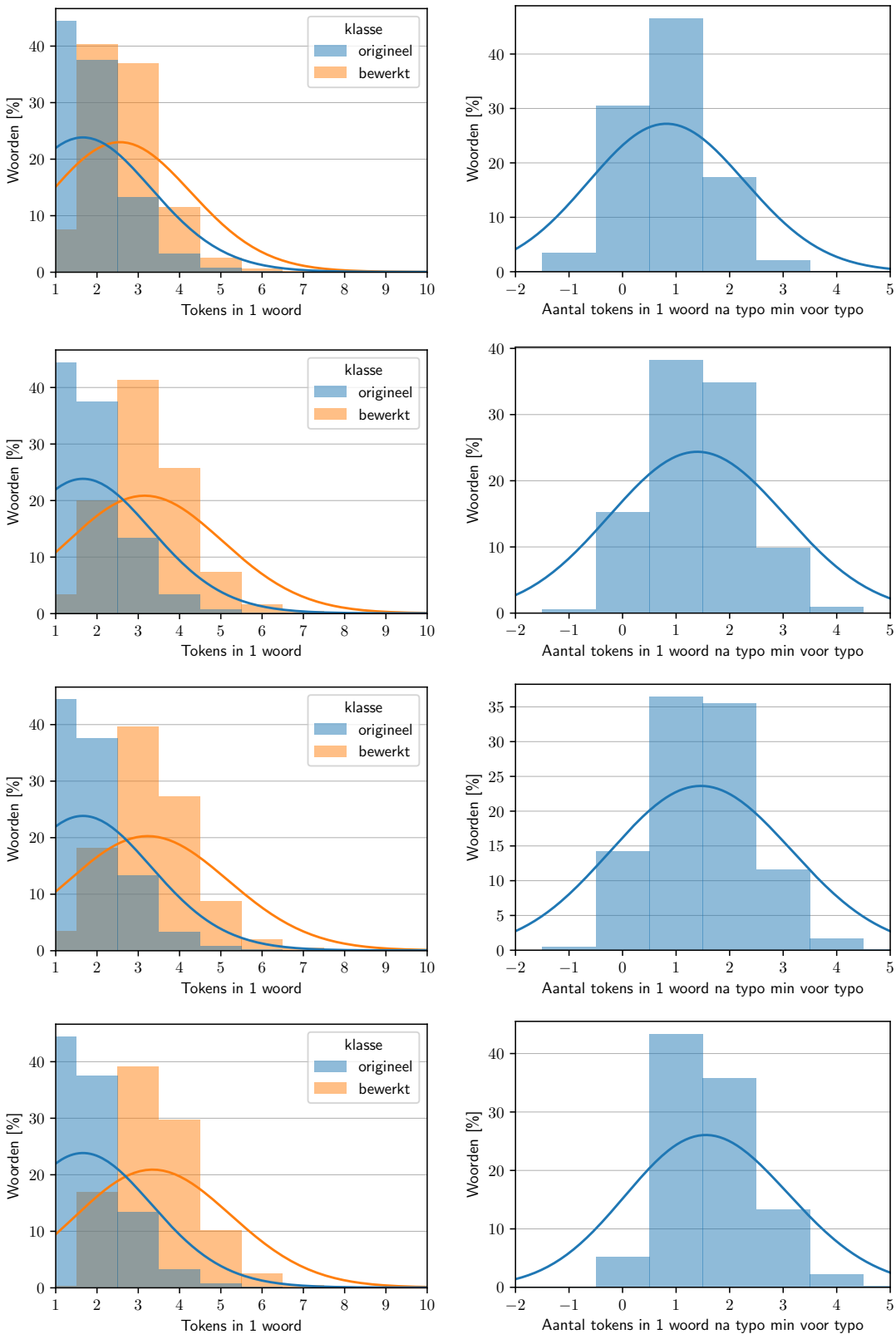


Figuur D.15 – Selectie uit de e-Lex-morfprefixboom met stamouder “po”.

D.8 Typostatistieken

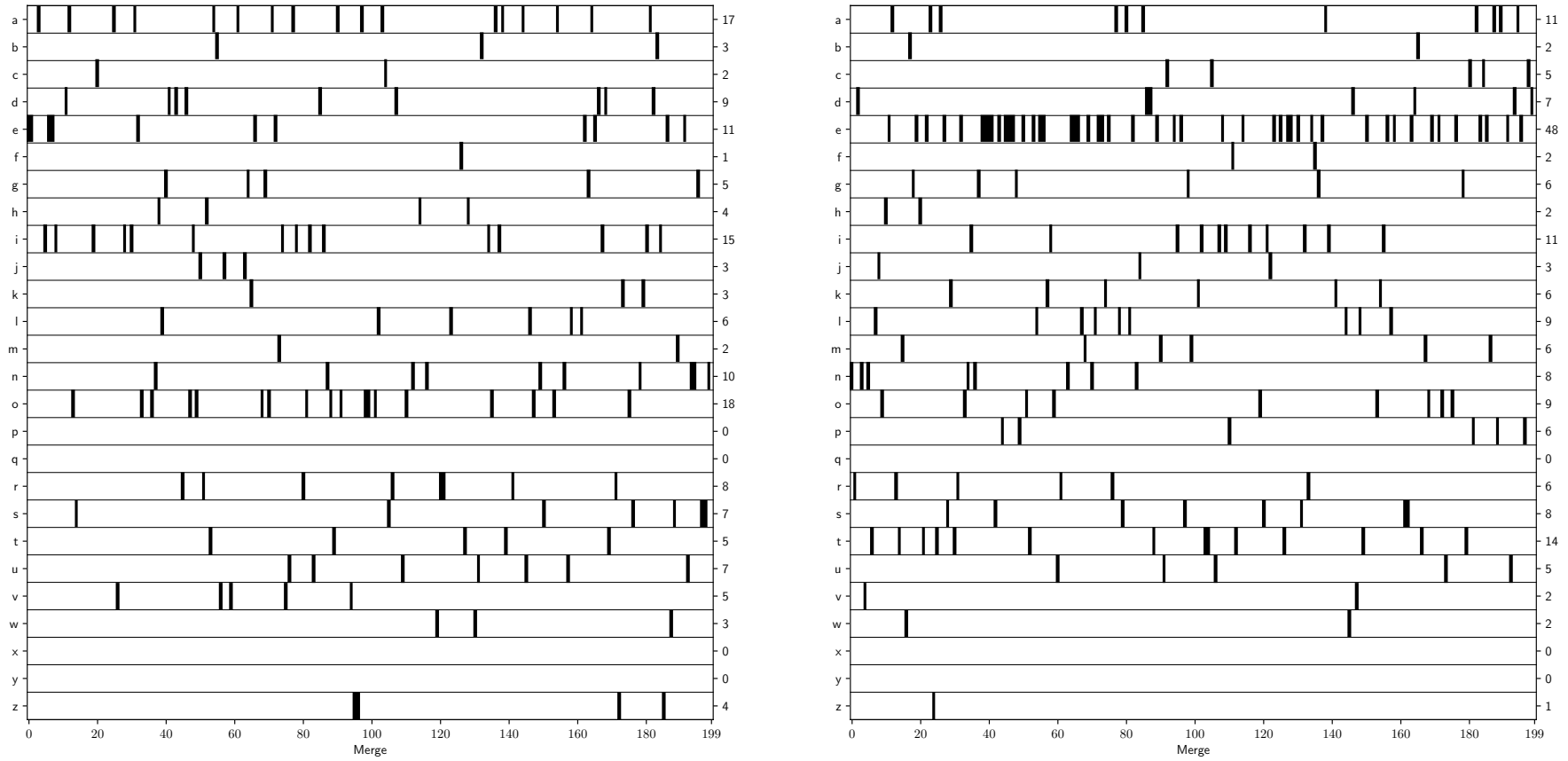


Figuur D.16 – Links: aantal BPE-tokens voor (blauw) en na (oranje) toevoegen van een typo in e-Lex-lemmata. Rechts: verschil voor hetzelfde lemma. Van boven naar onder: deletie, substitutie, wisseling, insertie.



Figuur D.17 – Links: aantal BPE-tokens voor (blauw) en na (oranje) toevoegen van een typo in NT2Lex-lemmata. Rechts: verschil voor hetzelfde lemma. Van boven naar onder: deletie, substitutie, wisseling, insertie.

D.9 Samenstellingen



Figuur D.18 – Karaktertijdlijn van de eerste 200 merges van BPE_{nl}. Links: de eindletter van het linkertype van elke merge. Rechts: de beginletter van het rechtertype van elke merge. Hoe meer streepjes een karakter links heeft, hoe liever het naar rechts wil mergen, en vice versa.

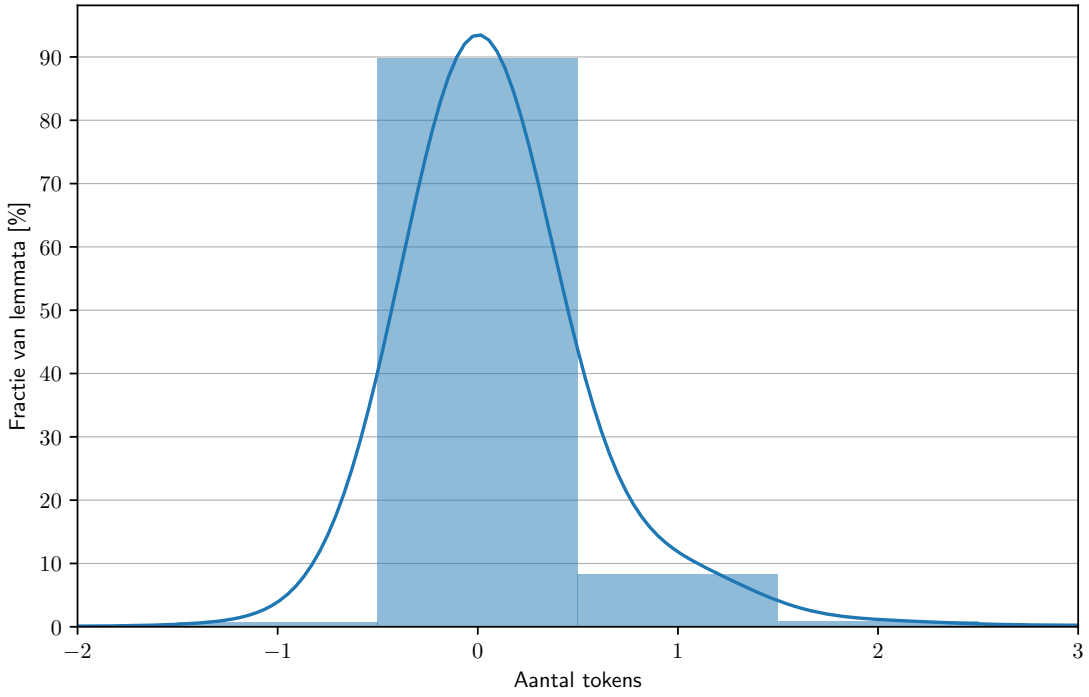
	Merge		Merge
1	e r	21	r e
2	e n	22	o e
3	i n	23	a t
4	e l	24	g e
5	s t	25	ch t
6	a a	26	i g
7	c h	27	o r
8	i e	28	a r
9	in g	29	ei d
10	i j	30	Ġ h
11	a n	31	l e
12	o n	32	Ġ m
13	o o	33	Ġ t
14	Ġ b	34	Ġ w
15	e i	35	aa r
16	Ġ s	36	r o
17	Ġ v	37	s ch
18	Ġ k	38	a l
19	u i	39	Ġ d
20	Ġ p	40	h eid

Tabel D.3 – Eerste 40 merges van een BPE-tokeniser die op de ongewogen morfen van e-Lex is getraind.

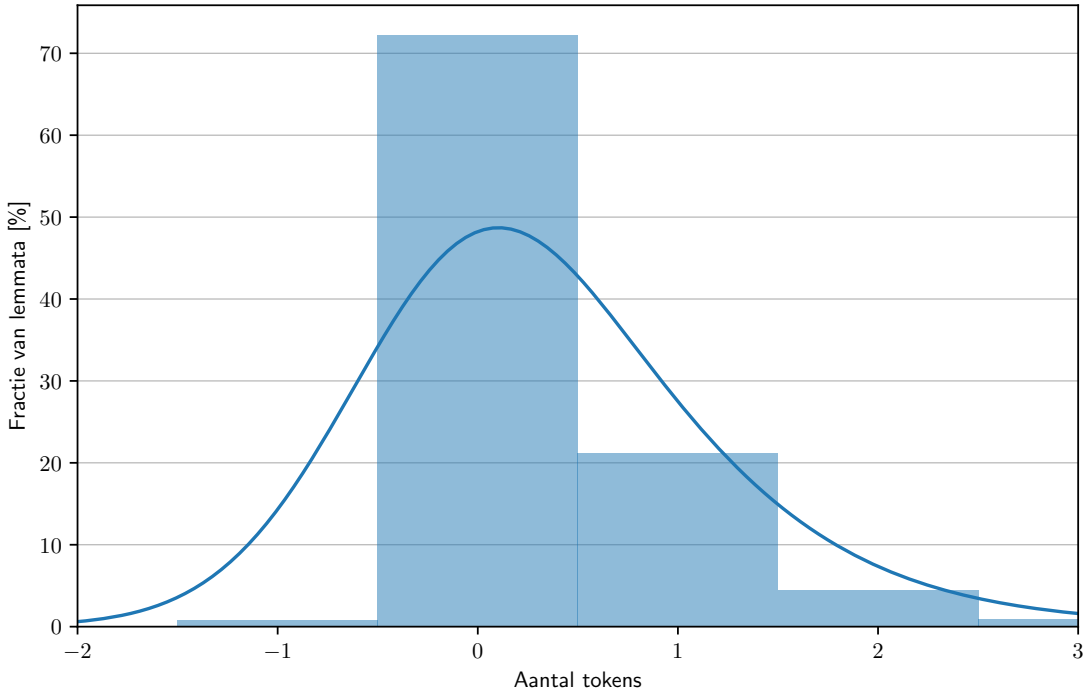
Woord	**bpe$_{nl}$-tokenisatie**
propaganda	Ġpropaganda
propagandabericht	Ġpropag and ab er icht
propagandablad	Ġpropag and abl ad
propagandaboodschap	Ġpropag and ab oodschap
propagandacampagne	Ġpropag and ac ampagne
propagandacompagnie	Ġpropag and ac omp agnie
propagandadepartement	Ġpropag and ade p artement
propagandadoeleinde	Ġpropag and ado ele inde
propagandafilm	Ġpropag and af ilm
propagandageschrift	Ġpropag and age schrift
propagandakosten	Ġpropag and ak osten
propagandaleugen	Ġpropag and ale ugen
propagandaliteratuur	Ġpropag and al iteratuur
propagandamachine	Ġpropag and am achine
propagandamachinerie	Ġpropag and am ach in erie
propagandamateriaal	Ġpropag and am ateriaal
propagandaministerie	Ġpropag and amin isterie
propagandanummer	Ġpropag and an ummer
propagandatechniek	Ġpropag and ate chniek
propagandatocht	Ġpropag and at ocht
propagandavoerder	Ġpropag and av oer der

Tabel D.4 – Affiniteit van de eindletter *a* voor rechtswaartse merges.

D.10 Knockout

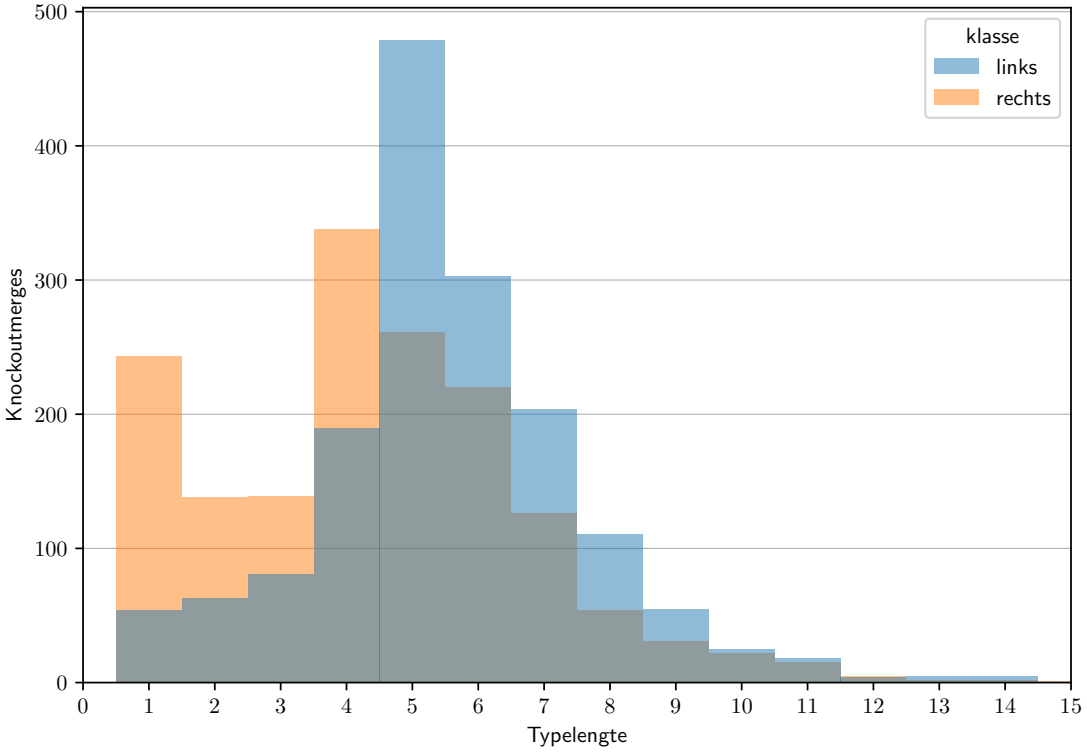


(a) Knockout via lexemische splitsingen

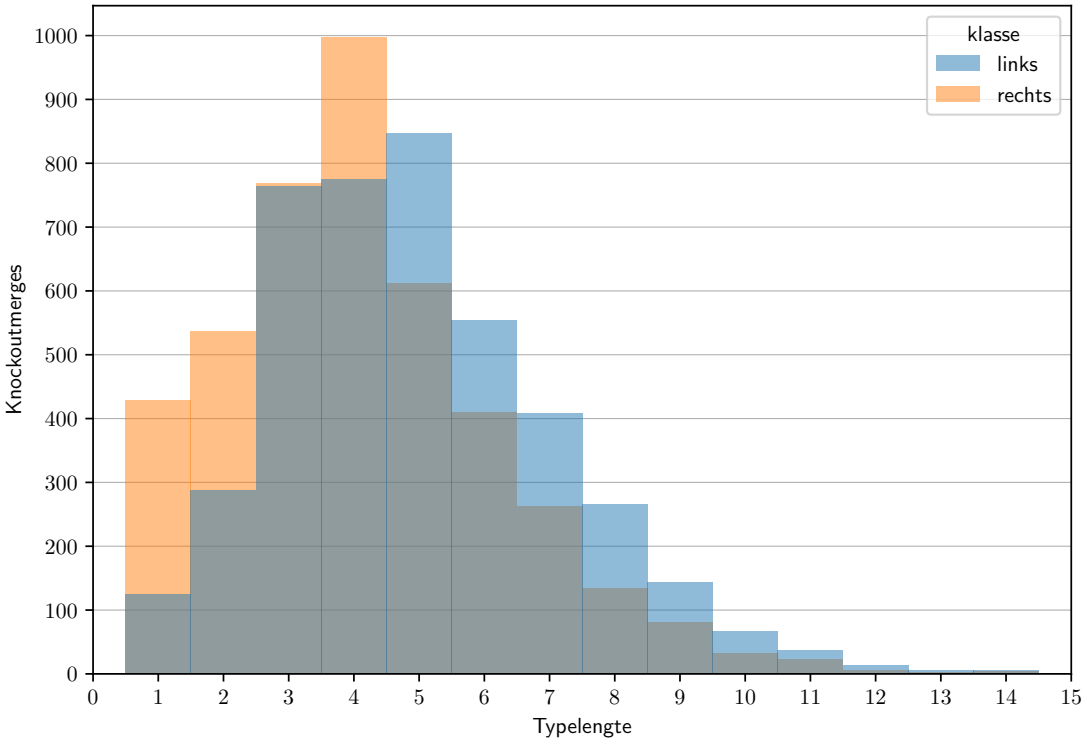


(b) Knockout via morfemische splitsingen

Figuur D.19 – Toename in aantal tokens bij tokenisatie van lemmata in e-Lex met BPE_{nl} na 50%-knockout.

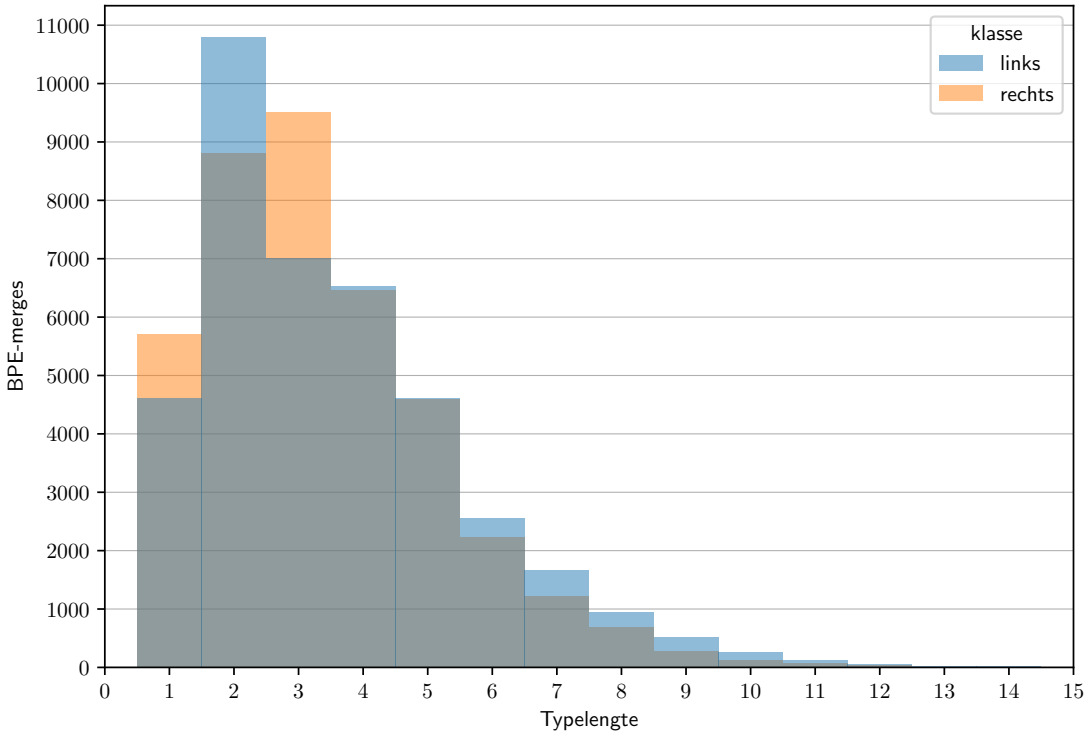


(a) Knockout via lexemische splitsingen

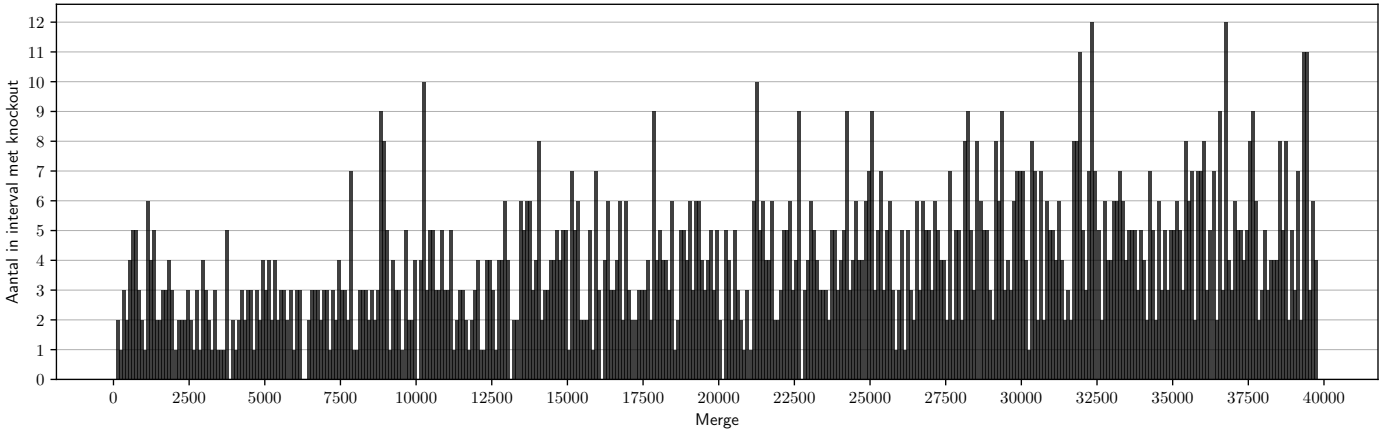


(b) Knockout via morfemische splitsingen

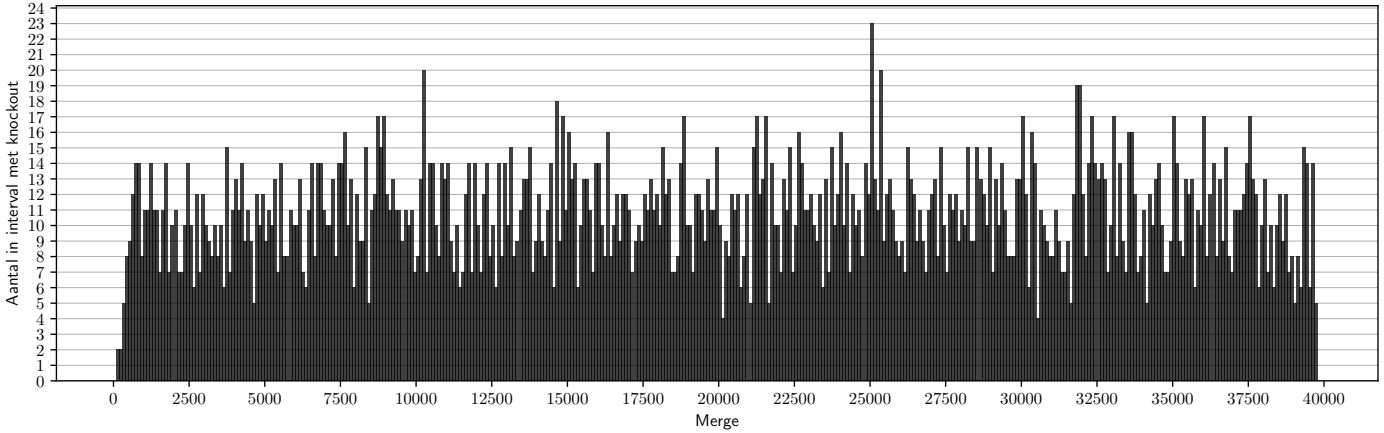
Figuur D.20 – Lengte van het linker- en rechtertype van verwijderde merges door 50%-knockout.



Figuur D.21 – Lengte van het linker- en rechtertype van alle merges in BPE_{nl}.



(a) Knockout via lexemische splitsingen



(b) Knockout via morfemische splitsingen

Figuur D.22 – Id's van schuldige merges bij 50%-knockout (gegroepeerd per 100 merges)

Bronvermelding

Referenties

Ács, Judit, Kádár, Ákos en Kornai, Andras (apr 2021). "Subword Pooling Makes a Difference". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, p. 2284–2295. DOI: 10.18653/v1/2021.eacl-main.194. URL: https://aclanthology.org/2021.eacl-main.194 (bezocht op 22-10-2022).

Amrhein, Chantal en Sennrich, Rico (nov 2021). "How Suitable Are Subword Segmentation Strategies for Translating Non-Concatenative Morphology?" In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Punta Cana, Dominican Republic: Association for Computational Linguistics, p. 689–705. DOI: 10.18653/v1/2021.findings-emnlp.60. URL: https://aclanthology.org/2021.findings-emnlp.60 (bezocht op 13-11-2022).

Ataman, Duygu en Federico, Marcello (mrt 2018). "An Evaluation of Two Vocabulary Reduction Methods for Neural Machine Translation". In: *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*. Boston, MA: Association for Machine Translation in the Americas, p. 97–110. URL: https://aclanthology.org/W18-1810 (bezocht op 18-12-2022).

Ataman, Duygu, Negri, Matteo, Turchi, Marco en Federico, Marcello (jun 2017). "Linguistically Motivated Vocabulary Reduction for Neural Machine Translation from Turkish to English". In: *The Prague Bulletin of Mathematical Linguistics* 108.1, p. 331–342. ISSN: 1804-0462. DOI: 10.1515/pralin-2017-0031. URL: http://archive.sciendo.com/PRALIN/pralin.2017.108.issue-1/pralin-2017-0031/pralin-2017-0031.pdf (bezocht op 18-12-2022).

Bahdanau, Dzmitry, Cho, Kyunghyun en Bengio, Yoshua (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: arXiv:1409.0473 [cs, stat] version: 7. ICLR. DOI: 10.48550/arXiv.1409.0473. URL: http://arxiv.org/abs/1409.0473 (bezocht op 24-02-2023).

Banerjee, Tamali en Bhattacharyya, Pushpak (jun 2018). "Meaningless yet meaningful: Morphology grounded subword-level NMT". In: *Proceedings of the Second Workshop on Subword/Character LEvel Models*. New Orleans: Association for Computational Linguistics, p. 55–60. DOI: 10.18653/v1/W18-1207. URL: https://aclanthology.org/W18-1207 (bezocht op 04-12-2022).

Beltagy, Iz, Lo, Kyle en Cohan, Arman (nov 2019). "SciBERT: A Pretrained Language Model for Scientific Text". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association

for Computational Linguistics, p. 3615–3620. DOI: 10.18653/v1/D19-1371. URL: https://aclanthology.org/D19-1371 (bezocht op 15-05-2023).

Bertels, Ann (jun 2022). "Chapter 14. Terminology and distributional analysis of corpora". In: *Theoretical Perspectives on Terminology: Explaining terms, concepts and specialized knowledge*. Red. door Pamela Faber en Marie-Claude L'Homme. Terminology and Lexicography Research and Practice. John Benjamins Publishing Company, p. 311–328. ISBN: 978-90-272-1106-4. DOI: 10.1075/tlrp.23.14ber. URL: https://benjamins.com/catalog/tlrp.23.14ber (bezocht op 29-05-2023).

Berton, A., Fetter, P. en Regel-Brietzmann, P. (okt 1996). "Compound words in large-vocabulary German speech recognition systems". In: *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*. Deel 2, 1165–1168 vol.2. DOI: 10.1109/ICSLP.1996.607814.

Bostrom, Kaj en Durrett, Greg (nov 2020). "Byte Pair Encoding is Suboptimal for Language Model Pretraining". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, p. 4617–4624. DOI: 10.18653/v1/2020.findings-emnlp.414. URL: https://aclanthology.org/2020.findings-emnlp.414 (bezocht op 21-11-2022).

Britz, Denny, Goldie, Anna, Luong, Minh-Thang en Le, Quoc (sep 2017). "Massive Exploration of Neural Machine Translation Architectures". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, p. 1442–1451. DOI: 10.18653/v1/D17-1151. URL: https://aclanthology.org/D17-1151 (bezocht op 05-02-2023).

Brown, Tom B., Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, Agarwal, Sandhini, Herbert-Voss, Ariel, Krueger, Gretchen, Henighan, Tom, Child, Rewon, Ramesh, Aditya, Ziegler, Daniel M., Wu, Jeffrey, Winter, Clemens, Hesse, Christopher, Chen, Mark, Sigler, Eric, Litwin, Mateusz, Gray, Scott, Chess, Benjamin, Clark, Jack, Berner, Christopher, McCandlish, Sam, Radford, Alec, Sutskever, Ilya en Amodei, Dario (jul 2020). *Language Models are Few-Shot Learners*. arXiv:2005.14165 [cs]. DOI: 10.48550/arXiv.2005.14165. URL: http://arxiv.org/abs/2005.14165 (bezocht op 04-02-2023).

Brysbaert, Marc, Stevens, Michaël, Mandera, Paweł en Keuleers, Emmanuel (2016). "How Many Words Do We Know? Practical Estimates of Vocabulary Size Dependent on Word Definition, the Degree of Language Input and the Participant's Age". In: *Frontiers in Psychology* 7. ISSN: 1664-1078. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2016.01116 (bezocht op 15-05-2023).

Bubeck, Sébastien, Chandrasekaran, Varun, Eldan, Ronen, Gehrke, Johannes, Horvitz, Eric, Kamar, Ece, Lee, Peter, Lee, Yin Tat, Li, Yuanzhi, Lundberg, Scott, Nori, Harsha, Palangi, Hamid, Ribeiro, Marco Tulio en Zhang, Yi (apr 2023). *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. arXiv:2303.12712 [cs]. DOI: 10.48550/arXiv.2303.12712. URL: http://arxiv.org/abs/2303.12712 (bezocht op 25-04-2023).

Carstairs-McCarthy, Andrew (jul 2008). "Lexeme, word-form, paradigm". In: *Lexeme, word-form, paradigm*. De Gruyter Mouton, p. 595–607. ISBN: 978-3-11-019401-2. DOI: 10.1515/9783110111286.1.9.595. URL: https://www.degruyter.com/document/doi/10.1515/9783110111286.1.9.595/html (bezocht op 11-12-2022).

Casas, Noe, Costa-jussà, Marta R. en Fonollosa, José A. R. (jul 2020). "Combining Subword Representations into Word-level Representations in the Transformer Archi-

tecture". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Online: Association for Computational Linguistics, p. 66–71. DOI: 10.18653/v1/2020.acl-srw.10. URL: https://aclanthology.org/2020.acl-srw.10 (bezocht op 22-10-2022).

Chitnis, Rohan en DeNero, John (sep 2015). "Variable-Length Word Encodings for Neural Translation Models". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, p. 2088–2093. DOI: 10.18653/v1/D15-1249. URL: https://aclanthology.org/D15-1249 (bezocht op 04-02-2023).

Chung, Hyung Won, Fevry, Thibault, Tsai, Henry, Johnson, Melvin en Ruder, Sebastian (2021). "Rethinking Embedding Coupling in Pre-trained Language Models". In: URL: https://openreview.net/forum?id=xpFFI_NtgpW (bezocht op 27-02-2023).

Chung, Hyung Won, Garrette, Dan, Tan, Kiat Chuan en Riesa, Jason (nov 2020). "Improving Multilingual Models with Language-Clustered Vocabularies". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, p. 4536–4546. DOI: 10.18653/v1/2020.emnlp-main.367. URL: https://aclanthology.org/2020.emnlp-main.367 (bezocht op 06-05-2023).

Clark, Jonathan H., Garrette, Dan, Turc, Iulia en Wieting, John (2022). "Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation". In: *Transactions of the Association for Computational Linguistics* 10. Place: Cambridge, MA Publisher: MIT Press, p. 73–91. DOI: 10.1162/tacl_a_00448. URL: https://aclanthology.org/2022.tacl-1.5 (bezocht op 21-11-2022).

Creutz, Mathias (jul 2003). "Unsupervised Segmentation of Words Using Prior Distributions of Morph Length and Frequency". In: *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. Sapporo, Japan: Association for Computational Linguistics, p. 280–287. DOI: 10.3115/1075096.1075132. URL: https://aclanthology.org/P03-1036 (bezocht op 09-04-2023).

Creutz, Mathias en Lagus, Krista (jul 2002). "Unsupervised Discovery of Morphemes". In: *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*. Association for Computational Linguistics, p. 21–30. DOI: 10.3115/1118647.1118650. URL: https://aclanthology.org/W02-0603 (bezocht op 01-04-2023).

Creutz, Mathias en Lagus, Krista (2005a). "Inducing the Morphological Lexicon of a Natural Language from Unannotated Text". In: *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning*. Deel 1, p. 51–59.

Creutz, Mathias en Lagus, Krista (mrt 2005b). *Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0*. technical A81. Helsinki University of Technology. URL: https://users.ics.aalto.fi/mcreutz/papers/Creutz05tr.pdf.

Creutz, Mathias en Lagus, Krista (feb 2007). "Unsupervised models for morpheme segmentation and morphology learning". In: *ACM Transactions on Speech and Language Processing* 4.1, 3:1–3:34. ISSN: 1550-4875. DOI: 10.1145/1187415.1187418. URL: https://dl.acm.org/doi/10.1145/1187415.1187418 (bezocht op 07-04-2023).

Creutz, Mathias en Lindén, Krister (2004). *Morpheme Segmentation Gold Standards for Finnish and English*. Tech. rap. A77. Helsinki University of Technology.

Dao, An, Kertkeidkachorn, Natthawut en Ichise, Ryutaro (jan 2021). "Comparison of Deep-Neural-Network-Based Models for Estimating Distributed Representations of Compound Words". In: *Procedia Computer Science*. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES2021 192, p. 1294–1303. ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.08.133. URL: https://www.sciencedirect.com/science/article/pii/S1877050921016227 (bezocht op 01-11-2022).

Delobelle, Pieter, Winters, Thomas en Berendt, Bettina (2020). "RobBERT: a Dutch RoBERTa-based Language Model". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, p. 3255–3265. DOI: 10.18653/v1/2020.findings-emnlp.292. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.292 (bezocht op 22-10-2022).

Delobelle, Pieter, Winters, Thomas en Berendt, Bettina (nov 2022). *RobBERT-2022: Updating a Dutch Language Model to Account for Evolving Language Use*. arXiv:2211.08192 [cs]. DOI: 10.48550/arXiv.2211.08192. URL: http://arxiv.org/abs/2211.08192 (bezocht op 26-11-2022).

DeRose, Steven J. (1988). "Grammatical Category Disambiguation by Statistical Optimization". In: *Computational Linguistics* 14.1, p. 31–39. URL: https://aclanthology.org/J88-1003 (bezocht op 03-06-2023).

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton en Toutanova, Kristina (jun 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, p. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423 (bezocht op 16-10-2022).

Dima, Corina en Hinrichs, Erhard (apr 2015). "Automatic Noun Compound Interpretation using Deep Neural Networks and Word Embeddings". In: *Proceedings of the 11th International Conference on Computational Semantics*. London, UK: Association for Computational Linguistics, p. 173–183. URL: https://aclanthology.org/W15-0122 (bezocht op 31-10-2022).

Ding, Shuoyang, Renduchintala, Adithya en Duh, Kevin (aug 2019). "A Call for Prudent Choice of Subword Merge Operations in Neural Machine Translation". In: *Proceedings of Machine Translation Summit XVII: Research Track*. Dublin, Ireland: European Association for Machine Translation, p. 204–213. URL: https://aclanthology.org/W19-6620 (bezocht op 18-12-2022).

Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, Uszkoreit, Jakob en Houlsby, Neil (jun 2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv:2010.11929 [cs]. DOI: 10.48550/arXiv.2010.11929. URL: http://arxiv.org/abs/2010.11929 (bezocht op 30-05-2023).

Downey, C.m., Xia, Fei, Levow, Gina-Anne en Steinert-Threlkeld, Shane (jul 2022). "A Masked Segmental Language Model for Unsupervised Natural Language Segmentation". In: *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Seattle, Washington: Association for Computational Linguistics, p. 39–50. DOI: 10.18653/v1/2022.sigmorphon-1.5. URL: https://aclanthology.org/2022.sigmorphon-1.5 (bezocht op 12-05-2023).

El Boukkouri, Hicham, Ferret, Olivier, Lavergne, Thomas, Noji, Hiroshi, Zweigenbaum, Pierre en Tsujii, Jun'ichi (dec 2020). "CharacterBERT: Reconciling ELMo and BERT for Word-Level Open-Vocabulary Representations From Characters". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, p. 6903–6915. DOI: 10.18653/v1/2020.coling-main.609. URL: https://aclanthology.org/2020.coling-main.609 (bezocht op 05-05-2023).

Ethayarajh, Kawin (nov 2019). "How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, p. 55–65. DOI: 10.18653/v1/D19-1006. URL: https://aclanthology.org/D19-1006 (bezocht op 22-10-2022).

Feng, Haodi, Chen, Kang, Deng, Xiaotie en Zheng, Weimin (2004). "Accessor Variety Criteria for Chinese Word Extraction". In: *Computational Linguistics* 30.1, p. 75–93. DOI: 10.1162/089120104773633394. URL: https://aclanthology.org/J04-1004 (bezocht op 26-03-2023).

Fritzinger, Fabienne en Fraser, Alexander (jul 2010). "How to Avoid Burning Ducks: Combining Linguistic Analysis and Corpus Statistics for German Compound Processing". In: *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*. Uppsala, Sweden: Association for Computational Linguistics, p. 224–234. URL: https://aclanthology.org/W10-1734 (bezocht op 06-11-2022).

Gage, Philip (1994). "A New Algorithm for Data Compression". In: *C Users Journal* 12.2, p. 23–38. URL: http://www.derczynski.com/papers/archive/BPE_Gage.pdf.

Galassi, Andrea, Lippi, Marco en Torroni, Paolo (okt 2021). "Attention in Natural Language Processing". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10. Conference Name: IEEE Transactions on Neural Networks and Learning Systems, p. 4291–4308. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3019893.

Gowda, Thamme en May, Jonathan (nov 2020). "Finding the Optimal Vocabulary Size for Neural Machine Translation". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, p. 3955–3964. DOI: 10.18653/v1/2020.findings-emnlp.352. URL: https://aclanthology.org/2020.findings-emnlp.352 (bezocht op 03-12-2022).

Grave, Edouard, Sukhbaatar, Sainbayar, Bojanowski, Piotr en Joulin, Armand (jul 2019). "Training Hybrid Language Models by Marginalizing over Segmentations". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, p. 1477–1482. DOI: 10.18653/v1/P19-1143. URL: https://aclanthology.org/P19-1143 (bezocht op 12-05-2023).

Grönroos, Stig-Arne, Virpioja, Sami en Kurimo, Mikko (mei 2020). "Morfessor EM+Prune: Improved Subword Segmentation with Expectation Maximization and Pruning". In: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, p. 3944–3953. ISBN: 979-10-95546-34-4. URL: https://aclanthology.org/2020.lrec-1.486 (bezocht op 18-03-2023).

Grönroos, Stig-Arne, Virpioja, Sami, Smit, Peter en Kurimo, Mikko (aug 2014). "Morfessor FlatCat: An HMM-Based Method for Unsupervised and Semi-Supervised Learning of Morphology". In: *Proceedings of COLING 2014, the 25th International Conference*

on Computational Linguistics: Technical Papers. Dublin, Ireland: Dublin City University en Association for Computational Linguistics, p. 1177–1185. URL: https://aclanthology.org/C14-1111 (bezocht op 10-04-2023).

He, Xuanli, Haffari, Gholamreza en Norouzi, Mohammad (jul 2020). "Dynamic Programming Encoding for Subword Segmentation in Neural Machine Translation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, p. 3042–3051. DOI: 10.18653/v1/2020.acl-main.275. URL: https://aclanthology.org/2020.acl-main.275 (bezocht op 18-12-2022).

Hochreiter, Sepp en Schmidhuber, Jürgen (nov 1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, p. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (bezocht op 04-03-2023).

Huck, Matthias, Riess, Simon en Fraser, Alexander (sep 2017). "Target-side Word Segmentation Strategies for Neural Machine Translation". In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, p. 56–67. DOI: 10.18653/v1/W17-4706. URL: https://aclanthology.org/W17-4706 (bezocht op 27-11-2022).

Keuleers, Emmanuel, Stevens, Michaël, Mandera, Paweł en Brysbaert, Marc (aug 2015). "Word knowledge in the crowd: Measuring vocabulary size and word prevalence in a massive online experiment". In: *Quarterly Journal of Experimental Psychology* 68.8. Publisher: SAGE Publications, p. 1665–1692. ISSN: 1747-0218. DOI: 10.1080/17470218.2015.1022560. URL: https://doi.org/10.1080/17470218.2015.1022560 (bezocht op 15-05-2023).

Kit, Chunyu en Wilks, Yorick (1999). "Unsupervised Learning of Word Boundary with Description Length Gain". In: *EACL 1999: CoNLL-99 Computational Natural Language Learning*. URL: https://aclanthology.org/W99-0701 (bezocht op 26-03-2023).

Klein, Stav en Tsarfaty, Reut (jul 2020). "Getting the ##life out of living: How Adequate Are Word-Pieces for Modelling Complex Morphology?" In: *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Online: Association for Computational Linguistics, p. 204–209. DOI: 10.18653/v1/2020.sigmorphon-1.24. URL: https://aclanthology.org/2020.sigmorphon-1.24 (bezocht op 06-05-2023).

Koehn, Philipp en Knight, Kevin (apr 2003). "Empirical Methods for Compound Splitting". In: *10th Conference of the European Chapter of the Association for Computational Linguistics*. Budapest, Hungary: Association for Computational Linguistics. URL: https://aclanthology.org/E03-1076 (bezocht op 06-11-2022).

Krott, Andrea (2001). *Analogy in morphology: the selection of linking elements in Dutch compounds*. ISBN: 978-90-76203-11-9. URL: https://repository.ubn.ru.nl/handle/2066/146830 (bezocht op 19-11-2022).

Kudo, Taku (jul 2018). "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, p. 66–75. DOI: 10.18653/v1/P18-1007. URL: https://aclanthology.org/P18-1007 (bezocht op 21-11-2022).

Kumar, Dipesh en Thawani, Avijit (mei 2022). “BPE beyond Word Boundary: How NOT to use Multi Word Expressions in Neural Machine Translation”. In: *Proceedings of the Third Workshop on Insights from Negative Results in NLP*. Dublin, Ireland: Association for Computational Linguistics, p. 172–179. DOI: 10.18653/v1/2022.insights-1.24. URL: https://aclanthology.org/2022.insights-1.24 (bezocht op 03-12-2022).

Kurimo, Mikko, Creutz, Mathias, Varjokallio, Matti, Arisoy, Ebru en Saraclar, Murat (jan 2006). “Unsupervised segmentation of words into morphemes – Challenge 2005 An Introduction and Evaluation Report”. In: *Proceedings of the PASCAL Challenge Workshop on Unsupervised segmentation of words into morphemes*, p. 1–11. URL: https://users.ics.aalto.fi/krista/papers/Kurimo06MorphoChallenge.pdf.

Lample, Guillaume en Conneau, Alexis (jan 2019). *Cross-lingual Language Model Pretraining*. arXiv:1901.07291 [cs]. DOI: 10.48550/arXiv.1901.07291. URL: http://arxiv.org/abs/1901.07291 (bezocht op 15-02-2023).

Laureys, Tom, De Pauw, Guy, Van hamme, Hugo, Daelemans, Walter en Van Compernolle, Dirk (mei 2004). “Evaluation and Adaptation of the Celex Dutch Morphological Database”. In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*. Lisbon, Portugal: European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2004/pdf/421.pdf (bezocht op 30-04-2023).

Lee, Jinhyuk, Yoon, Wonjin, Kim, Sungdong, Kim, Donghyeon, Kim, Sunkyu, So, Chan Ho en Kang, Jaewoo (feb 2020). “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”. In: *Bioinformatics* 36.4, p. 1234–1240. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btz682. URL: https://doi.org/10.1093/bioinformatics/btz682 (bezocht op 15-05-2023).

Liu, Yinhan, Ott, Myle, Goyal, Naman, Du, Jingfei, Joshi, Mandar, Chen, Danqi, Levy, Omer, Lewis, Mike, Zettlemoyer, Luke en Stoyanov, Veselin (jul 2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv:1907.11692 [cs]. URL: http://arxiv.org/abs/1907.11692 (bezocht op 16-10-2022).

Lui, Marco, Lau, Jey Han en Baldwin, Timothy (2014). “Automatic Detection and Language Identification of Multilingual Documents”. In: *Transactions of the Association for Computational Linguistics* 2. Place: Cambridge, MA Publisher: MIT Press, p. 27–40. DOI: 10.1162/tacl_a_00163. URL: https://aclanthology.org/Q14-1003 (bezocht op 23-05-2023).

Luo, Ziyang (2021). “Analyzing the Anisotropy Phenomenon in Transformer-based Masked Language Models”. Proefschrift. Uppsala University, Department of Linguistics en Philology.

Macken, Lieve en Tezcan, Arda (jul 2018). *Dutch compound splitting for bilingual terminology extraction*. Red. door Ruslan Mitkov, Johanna Monti, Gloria Corpas Pastor en Violeta Seretan. Current Issues in Linguistic Theory. Pages: 147-162 Publication Title: Multiword Units in Machine Translation and Translation Technology. John Benjamins Publishing Company. ISBN: 978-90-272-6420-6. DOI: 10.1075/cilt.341.07mac. URL: https://www-jbe-platform-com.kuleuven.e-bronnen.be/content/books/9789027264206 (bezocht op 05-11-2022).

Meng, Yu, Huang, Jiaxin, Wang, Guangyuan, Zhang, Chao, Zhuang, Honglei, Kaplan, Lance en Han, Jiawei (dec 2019). “Spherical text embedding”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 737. Red Hook, NY, USA: Curran Associates Inc., p. 8208–8217. (Bezocht op 16-10-2022).

Metz, Cade (nov 2020). "Meet GPT-3. It Has Learned to Code (and Blog and Argue)." In: *The New York Times*. ISSN: 0362-4331. URL: https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html (bezocht op 27-02-2023).

Mielke, Sebastian J., Alyafeai, Zaid, Salesky, Elizabeth, Raffel, Colin, Dey, Manan, Gallé, Matthias, Raja, Arun, Si, Chenglei, Lee, Wilson Y., Sagot, Benoît en Tan, Samson (dec 2021). *Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP*. arXiv:2112.10508 [cs]. DOI: 10.48550/arXiv.2112.10508. URL: http://arxiv.org/abs/2112.10508 (bezocht op 11-12-2022).

Mikolov, Tomas, Chen, Kai, Corrado, Greg en Dean, Jeffrey (sep 2013a). *Efficient Estimation of Word Representations in Vector Space*. arXiv:1301.3781 [cs]. URL: http://arxiv.org/abs/1301.3781 (bezocht op 20-03-2023).

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg en Dean, Jeffrey (dec 2013b). "Distributed representations of words and phrases and their compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., p. 3111–3119. (Bezocht op 29-10-2022).

Nguyen, Dat Quoc, Vu, Thanh en Tuan Nguyen, Anh (okt 2020). "BERTweet: A pre-trained language model for English Tweets". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, p. 9–14. DOI: 10.18653/v1/2020.emnlp-demos.2. URL: https://aclanthology.org/2020.emnlp-demos.2 (bezocht op 25-04-2023).

Ouyang, Long, Wu, Jeff, Jiang, Xu, Almeida, Diogo, Wainwright, Carroll L., Mishkin, Pamela, Zhang, Chong, Agarwal, Sandhini, Slama, Katarina, Ray, Alex, Schulman, John, Hilton, Jacob, Kelton, Fraser, Miller, Luke, Simens, Maddie, Askell, Amanda, Welinder, Peter, Christiano, Paul, Leike, Jan en Lowe, Ryan (mrt 2022). *Training language models to follow instructions with human feedback*. arXiv:2203.02155 [cs]. DOI: 10.48550/arXiv.2203.02155. URL: https://cdn.openai.com/papers/Training_language_models_to_follow_instructions_with_human_feedback.pdf (bezocht op 04-02-2023).

Papineni, Kishore, Roukos, Salim, Ward, Todd en Zhu, Wei-Jing (jul 2002). "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, p. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040 (bezocht op 06-03-2023).

Pennington, Jeffrey, Socher, Richard en Manning, Christopher (okt 2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, p. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (bezocht op 07-06-2023).

Post, Matt (okt 2018). "A Call for Clarity in Reporting BLEU Scores". In: *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, p. 186–191. DOI: 10.18653/v1/W18-6319. URL: https://aclanthology.org/W18-6319 (bezocht op 06-03-2023).

Provilkov, Ivan, Emelianenko, Dmitrii en Voita, Elena (jul 2020). "BPE-Dropout: Simple and Effective Subword Regularization". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational

Linguistics, p. 1882–1892. DOI: 10.18653/v1/2020.acl-main.170. URL: https://aclanthology.org/2020.acl-main.170 (bezocht op 19-11-2022).

Radford, Alec, Wu, Jeffrey, Child, Rewon, Luan, David, Amodei, Dario en Sutskever, Ilya (2019). "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog* 1.8. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (bezocht op 27-02-2023).

Raffel, Colin, Shazeer, Noam, Roberts, Adam, Lee, Katherine, Narang, Sharan, Matena, Michael, Zhou, Yanqi, Li, Wei en Liu, Peter J. (jul 2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv:1910.10683 [cs, stat]. URL: http://arxiv.org/abs/1910.10683 (bezocht op 15-10-2022).

Ramesh, Aditya, Dhariwal, Prafulla, Nichol, Alex, Chu, Casey en Chen, Mark (apr 2022). *Hierarchical Text-Conditional Image Generation with CLIP Latents*. arXiv:2204.06125 [cs]. DOI: 10.48550/arXiv.2204.06125. URL: https://cdn.openai.com/papers/dall-e-2.pdf (bezocht op 04-02-2023).

Reimers, Nils en Gurevych, Iryna (nov 2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, p. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410 (bezocht op 22-10-2022).

Rogers, Anna, Kovaleva, Olga en Rumshisky, Anna (jan 2021). "A Primer in BERTology: What We Know About How BERT Works". In: *Transactions of the Association for Computational Linguistics* 8, p. 842–866. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00349. URL: https://doi.org/10.1162/tacl_a_00349 (bezocht op 22-10-2022).

Rust, Phillip, Lotz, Jonas F., Bugliarello, Emanuele, Salesky, Elizabeth, Lhoneux, Miryam de en Elliott, Desmond (apr 2023). *Language Modelling with Pixels*. arXiv:2207.06991 [cs]. DOI: 10.48550/arXiv.2207.06991. URL: http://arxiv.org/abs/2207.06991 (bezocht op 30-05-2023).

Sahlgren, Magnus (2008). "The Distributional Hypothesis". In: *Rivista di Linguistica* 20.1, p. 33–53.

Salesky, Elizabeth, Runge, Andrew, Coda, Alex, Niehues, Jan en Neubig, Graham (okt 2018). *Optimizing Segmentation Granularity for Neural Machine Translation*. arXiv:1810.08641 [cs]. URL: http://arxiv.org/abs/1810.08641 (bezocht op 20-02-2023).

Schuster, Mike en Nakajima, Kaisuke (mrt 2012). "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X, p. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

Sennrich, Rico, Haddow, Barry en Birch, Alexandra (aug 2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, p. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://aclanthology.org/P16-1162 (bezocht op 07-11-2022).

Shwartz, Vered en Dagan, Ido (2019). "Still a Pain in the Neck: Evaluating Text Representations on Lexical Composition". In: *Transactions of the Association for Computational Linguistics* 7. Place: Cambridge, MA Publisher: MIT Press, p. 403–419. DOI: 10.1162/tacl_a_00277. URL: https://aclanthology.org/Q19-1027 (bezocht op 29-10-2022).

Shwartz, Vered en Waterson, Chris (jun 2018). "Olive Oil is Made *of* Olives, Baby Oil is Made *for* Babies: Interpreting Noun Compounds Using Paraphrases in a Neural Model". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, p. 218–224. DOI: 10.18653/v1/N18-2035. URL: https://aclanthology.org/N18-2035 (bezocht op 01-11-2022).

Sia, Suzanna, Dalmia, Ayush en Mielke, Sebastian J. (nov 2020). "Tired of Topic Models? Clusters of Pretrained Word Embeddings Make for Fast and Good Topics too!" In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, p. 1728–1736. DOI: 10.18653/v1/2020.emnlp-main.135. URL: https://aclanthology.org/2020.emnlp-main.135 (bezocht op 16-10-2022).

Stahlberg, Felix (okt 2020). "Neural Machine Translation: A Review". In: *Journal of Artificial Intelligence Research* 69, p. 343–418. ISSN: 1076-9757. DOI: 10.1613/jair.1.12007. URL: https://www.jair.org/index.php/jair/article/view/12007 (bezocht op 19-05-2023).

Stymne, Sara (2008). "German Compounds in Factored Statistical Machine Translation". In: *Advances in Natural Language Processing*. Red. door Bengt Nordström en Aarne Ranta. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, p. 464–475. ISBN: 978-3-540-85287-2. DOI: 10.1007/978-3-540-85287-2_44.

Suárez, Pedro Javier Ortiz, Sagot, Benoît en Romary, Laurent (jul 2019). "Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures". In: Leibniz-Institut für Deutsche Sprache. DOI: 10.14618/IDS-PUB-9021. URL: https://inria.hal.science/hal-02148693 (bezocht op 12-05-2023).

Sun, Zhiqing en Deng, Zhi-Hong (okt 2018). "Unsupervised Neural Word Segmentation for Chinese via Segmental Language Modeling". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, p. 4915–4920. DOI: 10.18653/v1/D18-1531. URL: https://aclanthology.org/D18-1531 (bezocht op 12-05-2023).

Sutskever, Ilya, Vinyals, Oriol en Le, Quoc V (2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. Deel 27. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html (bezocht op 04-03-2023).

Taalunie (2014). *e-Lex 1.1.1 voor Taal- en Spraaktechnologie*. Tech. rap. NTU/NWO: Instituut voor de Nederlandse Taal. URL: http://hdl.handle.net/10032/tm-a2-h2 (bezocht op 14-04-2023).

Tack, Anaïs (2021). "Mark my words! On the automated prediction of lexical difficulty for foreign language readers". Proefschrift. UCL - Université Catholique de Louvain. URL: https://dial.uclouvain.be/pr/boreal/object/boreal:248811 (bezocht op 16-10-2022).

Tack, Anaïs, François, Thomas, Desmet, Piet en Fairon, Cédrick (2018). "NT2Lex: A CEFR-Graded Lexical Resource for Dutch as a Foreign Language Linked to Open Dutch WordNet". In: *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*. New Orleans, Louisiana: Association for Computational Linguistics, p. 137–146. DOI: 10.18653/v1/W18-0514. URL: http://aclweb.org/anthology/W18-0514 (bezocht op 16-10-2022).

Tai, Wen, Kung, H. T., Dong, Xin, Comiter, Marcus en Kuo, Chang-Fu (nov 2020). "exBERT: Extending Pre-trained Models with Domain-specific Vocabulary Under Constrained Training Resources". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, p. 1433–1439. DOI: 10.18653/v1/2020.findings-emnlp.129. URL: https://aclanthology.org/2020.findings-emnlp.129 (bezocht op 25-04-2023).

Tamchyna, Aleš, Weller-Di Marco, Marion en Fraser, Alexander (sep 2017). "Modeling Target-Side Inflection in Neural Machine Translation". In: *Proceedings of the Second Conference on Machine Translation*. Copenhagen, Denmark: Association for Computational Linguistics, p. 32–42. DOI: 10.18653/v1/W17-4704. URL: https://aclanthology.org/W17-4704 (bezocht op 10-12-2022).

Tay, Yi, Tran, Vinh Q., Ruder, Sebastian, Gupta, Jai, Chung, Hyung Won, Bahri, Dara, Qin, Zhen, Baumgartner, Simon, Yu, Cong en Metzler, Donald (mrt 2022). "Charformer: Fast Character Transformers via Gradient-based Subword Tokenization". In: URL: https://openreview.net/forum?id=JtBRnrlOEFN (bezocht op 20-11-2022).

Thompson, Laure en Mimno, David (okt 2020). "Topic Modeling with Contextualized Word Representation Clusters". In: DOI: 10.48550/arXiv.2010.12626. URL: https://arxiv.org/abs/2010.12626v1 (bezocht op 17-10-2022).

Unicode Consortium (2022). *The Unicode Standard*. Tech. rap. Version 15.0.0. ISBN 978-1-936213-32-0. The Unicode Consortium, p. 1060. URL: https://www.unicode.org/versions/Unicode15.0.0/UnicodeStandard-15.0.pdf (bezocht op 27-02-2023).

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz en Polosukhin, Illia (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Deel 30. Curran Associates, Inc. URL: https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (bezocht op 29-10-2022).

Verhoeven, Ben, Daelemans, Walter en Huyssteen, Gerhard B van (2012). "Classification of Noun-Noun Compound Semantics in Dutch and Afrikaans". In: *Proceedings of the Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*, p. 121–125. URL: https://repository.uantwerpen.be/docman/irua/e98d6a/104644.pdf (bezocht op 25-03-2023).

Vilar, David en Federico, Marcello (aug 2021). "A Statistical Extension of Byte-Pair Encoding". In: *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*. Bangkok, Thailand (online): Association for Computational Linguistics, p. 263–275. DOI: 10.18653/v1/2021.iwslt-1.31. URL: https://aclanthology.org/2021.iwslt-1.31 (bezocht op 03-12-2022).

Viterbi, A. (apr 1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13.2. Conference Name: IEEE Transactions on Information Theory, p. 260–269. ISSN: 1557-9654. DOI: 10.1109/TIT.1967.1054010.

Wang, Changhan, Cho, Kyunghyun en Gu, Jiatao (dec 2019). *Neural Machine Translation with Byte-Level Subwords*. arXiv:1909.03341 [cs]. URL: http://arxiv.org/abs/1909.03341 (bezocht op 27-02-2023).

Wang, Shufan, Thompson, Laure en Iyyer, Mohit (nov 2021). "Phrase-BERT: Improved Phrase Embeddings from BERT with an Application to Corpus Exploration". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online en Punta Cana, Dominican Republic: Association for Computati-

onal Linguistics, p. 10837–10851. DOI: 10.18653/v1/2021.emnlp-main.846. URL: https://aclanthology.org/2021.emnlp-main.846 (bezocht op 30-10-2022).

Weller, Marion, Cap, Fabienne, Müller, Stefan, Schulte im Walde, Sabine en Fraser, Alexander (aug 2014). "Distinguishing Degrees of Compositionality in Compound Splitting for Statistical Machine Translation". In: *Proceedings of the First Workshop on Computational Approaches to Compound Analysis (ComAComA 2014)*. Dublin, Ireland: Association for Computational Linguistics en Dublin City University, p. 81–90. DOI: 10.3115/v1/W14-5709. URL: https://aclanthology.org/W14-5709 (bezocht op 29-10-2022).

Wu, Yingting en Zhao, Hai (2018). "Finding Better Subword Segmentation for Neural Machine Translation". In: *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*. Red. door Maosong Sun, Ting Liu, Xiaojie Wang, Zhiyuan Liu en Yang Liu. Lecture Notes in Computer Science. Cham: Springer International Publishing, p. 53–64. ISBN: 978-3-030-01716-3. DOI: 10.1007/978-3-030-01716-3_5.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Łukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa, Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff en Dean, Jeffrey (okt 2016). *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv:1609.08144 [cs]. DOI: 10.48550/arXiv.1609.08144. URL: http://arxiv.org/abs/1609.08144 (bezocht op 07-04-2023).

Xu, Jingjing, Zhou, Hao, Gan, Chun, Zheng, Zaixiang en Li, Lei (aug 2021). "Vocabulary Learning via Optimal Transport for Neural Machine Translation". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, p. 7361–7373. DOI: 10.18653/v1/2021.acl-long.571. URL: https://aclanthology.org/2021.acl-long.571 (bezocht op 19-02-2023).

Xue, Linting, Barua, Aditya, Constant, Noah, Al-Rfou, Rami, Narang, Sharan, Kale, Mihir, Roberts, Adam en Raffel, Colin (2022). "ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models". In: *Transactions of the Association for Computational Linguistics* 10. Place: Cambridge, MA Publisher: MIT Press, p. 291–306. DOI: 10.1162/tacl_a_00461. URL: https://aclanthology.org/2022.tacl-1.17 (bezocht op 30-05-2023).

Yazdani, Majid, Farahmand, Meghdad en Henderson, James (sep 2015). "Learning Semantic Composition to Detect Non-compositionality of Multiword Expressions". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, p. 1733–1742. DOI: 10.18653/v1/D15-1201. URL: https://aclanthology.org/D15-1201 (bezocht op 12-11-2022).

Zanetti, Arianna (sep 2020). "NLP methods for the automatic generation of exercises for second language learning from parallel corpus data". In: Accepted: 2020-09-25T07:36:28Z. URL: https://gupea.ub.gu.se/handle/2077/66583 (bezocht op 22-10-2022).

Zhao, Hai en Kit, Chunyu (2008). “An Empirical Comparison of Goodness Measures for Unsupervised Chinese Word Segmentation with a Unified Framework”. In: *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*. URL: https://aclanthology.org/I08-1002 (bezocht op 26-03-2023).

Zhou, Giulio (2018). “Morphological Zero-Shot Neural Machine Translation”. Proefschrift. University of Edinburgh. URL: https://project-archive.inf.ed.ac.uk/msc/20183019/msc_proj.pdf.

Verdere bronnen

Araabi, Ali, Monz, Christof en Niculae, Vlad (sep 2022). “How Effective is Byte Pair Encoding for Out-Of-Vocabulary Words in Neural Machine Translation?” In: *Proceedings of the 15th biennial conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*. Orlando, USA: Association for Machine Translation in the Americas, p. 117–130. URL: https://aclanthology.org/2022.amta-research.9 (bezocht op 12-05-2023).

Creutz, Mathias en Lagus, Krista (jul 2004). “Induction of a Simple Morphology for Highly-Inflecting Languages”. In: *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*. Barcelona, Spain: Association for Computational Linguistics, p. 43–51. URL: https://aclanthology.org/W04-0106 (bezocht op 08-05-2023).

Nandakumar, Navnita, Baldwin, Timothy en Salehi, Bahar (jun 2019). “How Well Do Embedding Models Capture Non-compositionality? A View from Multiword Expressions”. In: *Proceedings of the 3rd Workshop on Evaluating Vector Space Representations for NLP*. Minneapolis, USA: Association for Computational Linguistics, p. 27–34. DOI: 10.18653/v1/W19-2004. URL: https://aclanthology.org/W19-2004 (bezocht op 01-11-2022).

Niessen, Sonja en Ney, Hermann (2001). “Toward hierarchical models for statistical machine translation of inflected languages”. In: *Proceedings of the ACL 2001 Workshop on Data-Driven Methods in Machine Translation*. URL: https://aclanthology.org/W01-1407 (bezocht op 07-02-2023).

Zhang, Zhuosheng, Wu, Yuwei, Zhao, Hai, Li, Zuchao, Zhang, Shuailiang, Zhou, Xi en Zhou, Xiang (apr 2020). “Semantics-Aware BERT for Language Understanding”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.05. Number: 05, p. 9628–9635. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i05.6510. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6510 (bezocht op 22-10-2022).

n

Index

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z