BUILDING AND EVALUATING OPEN-VOCABULARY LANGUAGE MODELS

by
Sebastian Jonathan Mielke

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
October 2023

Abstract

Language models have always been a fundamental NLP tool and application. This thesis focuses on *open-vocabulary language models*, i.e., models that can deal with novel and unknown words at runtime. We will propose both new ways to *construct* such models as well as *use* such models in cross-linguistic *evaluations* to answer questions of difficulty and language-specificity in modern NLP tools.

We start by surveying linguistic background as well as past and present NLP approaches to tokenization and open-vocabulary language modeling (Mielke et al., 2021). Thus equipped, we establish desirable principles for such models, both from an engineering mindset as well as a linguistic one and hypothesize a model based on the marriage of neural language modeling and Bayesian nonparametrics to handle a truly infinite vocabulary, boasting attractive theoretical properties and mathematical soundness, but presenting practical implementation difficulties. As a compromise, we thus introduce a word-based two-level language model that still has many desirable characteristics while being highly feasible to run (Mielke and Eisner, 2019). Unlike the more dominant approaches of characters or subword units as one-layer tokenization it uses words; its key feature is the ability to generate novel words in context and in isolation.

Moving on to evaluation, we ask: how do such models deal with the wide variety of languages of the world—are they struggling with some languages? Relating this question to a more linguistic one, are some languages inherently more difficult to deal with? Using simple methods, we show that indeed they are, starting with a small pilot study that suggests typological predictors of difficulty (Cotterell et al., 2018). Thus encouraged,

we design a far bigger study with more powerful methodology, a principled and highly feasible evaluation and comparison scheme based again on multi-text likelihood (Mielke et al., 2019). This larger study shows that the earlier conclusion of typological predictors is difficult to substantiate, but also offers a new insight on the complexity of Translationese. Following that theme, we end by extending this scheme to machine translation models to answer questions traditional evaluation metrics like BLEU cannot (Bugliarello et al., 2020).

Thesis Readers

Dr. Jason Eisner (Primary Advisor)
Professor
Department of Computer Science
Johns Hopkins University

Dr. Daniel Khashabi
Assistant Professor
Department of Computer Science
Johns Hopkins University

Dr. Matt Post
Adjunct Professor
Department of Computer Science
Johns Hopkins University

Dr. Brian Roark
Research Scientist
Google Research

Acknowledgements

As with every paper I wrote, let me first say that I am thankful to the anonymous reviewers that gave valuable feedback for polishing the papers this thesis is based on. Likewise, to get it out of the way: the research presented in Chapter 8 was supported by the National Science Foundation under Grant No. 1718846 and the research presented in Chapter 9 has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199, the National Science Foundation under grant 1761548, and by "Research and Development of Deep Learning Technology for Advanced Multilingual Speech Translation," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

Equally formulaic but no less important, all members of my committee deserve great thanks for sticking with me through the ups and downs, delays upon delays, all the way through especially this last year of 2023 and its adrenaline-fueled conclusion (I guess that is how these things always go).

My advisor Jason Eisner in particular shaped my life like very few other people did these past six years and change of my life. I came in a hungry little international student and I feel like for all of my first year our weekly meetings were mostly private lectures to indoctrinate me into the world of Bayesianism (and of proper typesetting for English language publications). For that and the tireless committment to making time for me and everyone else in the lab no matter the hour I am truly grateful.

I feel uniquely grateful to Emily Dinan and Y-Lan Boureau for the best internship of my life, and all the people at FAIR I interacted with during that internship in 2020—for showing me what a dream research environment might look like... and that such a thing can exist, to begin with!

The two papers in this thesis that do not bear my advisors name are two of the many I wrote with senior-labmate Ryan Cotterell, who for the first few years I considered Jason's counterpart in many important ways in my advising. Many nights spent in various beer-serving establishments taught me the dirtier secrets of success in the academic and NLP world and I have much of my career to thank him for.

Speaking of labmates, I would not be submitting this thesis without the many conversations and both technical, lab, and life advice from Tim Vieira, Chu-Cheng Lin, Hongyuan Mei, and Matthew Francis-Landau.

There are many students outside the lab but within CLSP that I shared similar conversations with and greatly miss, typing these paragraphs up in NYC with them all over the globe: Aaron Mueller, Adam Poliak, Annabelle Carrell, Anton Belyy, Chandler May, David Mueller, Elizabeth Salesky, Huda Khayrallah, Kelly St. Denis, Pamela Shapiro, Patrick Xia, Pushpendre Rastogi, Rachel Rudinger, Rebecca Knowles, Shuoyang Ding, Suzanna Sia, Tongfei Chen, Winston Wu, and Zach Wood-Doughty (some of whom I got to enjoy as the most excellent roommates I ever had!).

The NYC connections I made during my time in CLSP, all of which of course have ties to Hopkins, continue to be a blessing in my life up to this point: Lawrence Wolf-Sonkin, Naomi Saphra, Nikita Nangia, and Stephen Roller. Who better to turn to in dire need of food and company?

Now before this list degrades into listing all my favorite NLP people I got to befriend over the years (I would feel bad for forgetting about some, no doubt!), I'd like to mention the people I often look forward to most when I come down to Baltimore, people that are

not at all connected to CLSP and my research: Carlie Hruban and Margaret Eminizer in particular come to mind, both having changed my life with the smallest bits of advice and continuing to make it so much richer for hopefully a while to come—in the meantime I can only try to pay it forward.

And with that we come to the actual heart of this storm of names, the people who have been there for me on the many nights in which I found myself hyperventilating crying because everything was too much and I didn't see any light at the end of the tunnel.

Xiang Lorraine Li, who, always one year ahead of me, was a source of much guidance through the PhD from applications to job search.

Vagrant Gautam, who went from academic collaborator to my dearest and best friend and practically family, sticking with me through the ugliest lows of the past few years and growing together with me through the rollercoaster that is our academic career.

Claire Cramer, who gave me so much hope that better things were possible and never forced but always empowered me to overcome the struggles of the past two years.

Finally, I would like to thank alcohol and drugs for saving my life when all other avenues were exhausted—and the fellowships of Alcoholics Anonymous, Narcotics Anonymous, and Recovery Dharma with the many wonderful people who are just like me for saving it again when I was ready to try again.

Papers included in this thesis

Wolf-Sonkin et al. (2018)'s exposition is partially reproduced in Section 2.6.

Lawrence Wolf-Sonkin, Jason Naradowsky, Sabrina J. Mielke, and Ryan Cotterell. 2018. A structured variational autoencoder for contextual morphological inflection. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2631–2641, Melbourne, Australia. Association for Computational Linguistics.

The four shared first authors implemented various parts of the pipeline in isolation and all contributed to writing equally.

Mielke et al. (2021) is largely reproduced in Chapter 3.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. CoRR, abs/2112.10508.

The first author contributed the majority of writing with the remaining authors contributing sections and subsections with various degrees of editing by the first author.

Mielke and Eisner (2019) is largely reproduced in Chapter 6, with parts appearing in Section 2.5 and Chapter 4.

Sabrina J. Mielke and Jason Eisner. 2019. Spell once, summon anywhere: A two-level open-vocabulary language model. Proceedings of the AAAI Conference on Artificial

Intelligence, 33(01):6843–6850.

Cotterell et al. (2018) is largely reproduced in Chapter 7, with parts appearing in Chapter 4.

Ryan Cotterell, Sabrina J. Mielke, Jason Eisner, and Brian Roark. 2018. Are all languages equally hard to language-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), pages 536–541, New Orleans, Louisiana. Association for Computational Linguistics.

Modeling and analysis to a large degree was done by the first author, framing and writing were mostly jointly written by the first two authors.

Mielke et al. (2019) is largely reproduced in Chapter 8.

Sabrina J. Mielke, Ryan Cotterell, Kyle Gorman, Brian Roark, and Jason Eisner. 2019. What kind of language is hard to language-model? In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 4975–4989, Florence, Italy. Association for Computational Linguistics.

Modeling, analysis, and writing were mostly done by the first author with the third author specifically contributing the ADL analysis and the last four authors all providing advising.

Bugliarello et al. (2020) is largely reproduced in Chapter 9.

Emanuele Bugliarello, Sabrina J. Mielke, Antonios Anastasopoulos, Ryan Cotterell, and Naoaki Okazaki. 2020. It's easier to translate out of English than into it: Measuring neural translation difficulty by cross-mutual information. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 1640–1649, Online. Association for Computational Linguistics.

Modeling was done by the first author, framing and eventual definition of the metric and analysis by the second author, with the last three authors advising.

Table of Contents

List of Tables

List of Figures

Part I

Setting the stage

Chapter 1

Introduction

In 2023, it feels silly to explain what a language model is—after all, even the general public is keenly aware of recent advances, at the moment of this writing represented in OpenAI's ChatGPT, an "AI tool" that seems to be able to speak, think, and follow instructions creatively. Much has been written on claims like these, and before we even talk about what this thesis *is*, we should be clear on what it *isn't*: we will not spend time on billion-parameter large language models (LLMs) and their fascinating generative capabilities (though we will try to point out how much of this research may still apply and be useful for an LLM-dominated NLP field). Instead we will rewind the clock a few years to hone in on the now underappreciated underpinnings of these imposing poster children of AI: the concept of *open-vocabulary language models* in general.

After all, the concepts of language models are not new, as the idea of language models in particular could be claimed to be almost a hundred years old—all that *is* new is their impressive performance based on neural networks, powered by immense data and compute. The fundamental idea remains the same throughout: we want to assign probabilities to written text, say a single sentence (a more detailed definition of all that entails will follow in the background chapters of Part I).

A language model thus is a probability distribution over the infinite set of strings, and this implies two kinds of uses: one in which we *score* or *rank* strings to decide which one

is more likely (implying it is more natural in the given language), and one in which we *generate* strings by sampling from the distribution. The former was certainly the main if not exclusive use case for decades as models simply were not powerful enough to generate sensible text, but their scoring would prove useful in noisy-channel-like settings as used in the early NLP tasks of speech recognition and machine translation. The latter on the other hand, as described above has become the focus of much attention in recent years as models are powerful enough to generate not only coherent sentences, but even entire paragraphs and documents.

The research in this thesis sits right at the cusp between the two views of language models, both in terms of content as well as in terms of timeline, putting us into an interesting position to consider both. We will heavily use the probabilistic foundations of language models not just in our evaluation work but also in the definitions we will posit for new models, while also very much analyzing our models' generative outputs.

The qualifier "open-vocabulary" can be described simply as having a language model no longer be restricted to a fixed vocabulary of a finite set of words, but instead giving it the ability to generate any word in the language, even those not seen in training.

This greatly matters as text "in the wild" is full of such words: sometimes they are just misspellings, sometimes inflected word forms (especially in languages that are morphologically richer than English), and sometimes they are highly meaningful new words. As models become interesting for generation purposes, this concept has been accepted to be crucial, as earlier approaches of simply ignoring the content of unknown words (i.e., words not seen in training) may work for scoring sentences, but certainly not for successful generation. Advances like BPE (Sennrich et al., 2016) have made it easy if not trivial to turn language models as well as most other NLP models open-vocabulary, but as we will see, they are certainly not without their own issues. For that reason one big part of this thesis focuses on proposing new and perhaps smarter ways of being open-vocabulary in language modeling. The other big part of this thesis will take existing open-vocabulary

language models and compare them on a variety of the world's languages, asking whether they are equally good at modeling all languages or whether some languages are harder to model than others—and whether there is some systematicity to these differences.

In all this, it is worth noting that this work like earlier work on NLP strongly connects to ideas, principles, and questions from linguistics. As we will see in Sections 1.2 and 1.3, both main parts of this thesis will do so differently: Part II asks how open-vocab language modeling interacts with *fundamental ideas of word formation in linguistics* while Part III ask more closely how it relates to *linguistic typology*.

1.1 Part I: Background

Before we can dive into how we improve on existing understanding, we will have to establish background that is relevant to all happening in this thesis.

Chapter 2 will formally introduce the term "open-vocabulary language model," giving a brief history of language models and their uses without going into details of possible implementations pre- and post-neural era (which we postpone for Chapter 4). With this definition and some use cases already sketched, we will cover how language models tend to be evaluated and be clear on our focus on classic probabilistic evaluation, i.e., perplexity, again leaving interesting intricacies for later chapters on comparing perplexities (in particular Chapter 7). Rounding out this bird's-eye introduction to language models, we will also cover the usual way in which *neural*[1] language models are trained: stochastic gradient descent on a regularized likelihood loss, batching multiple strings of a reasonably long length (obtained by cutting the training data, seen as a single long string, into pieces). Finally, we will also venture a look at a linguistic perspective, particularly word formation through inflectional morphology, that is how the singular and plural forms of BOOK are *book* and *books* respectively, two "distinct" words that share the same stem as they are inflections

[1]What we discuss more broadly applies to all models trained with gradient descent on a log-likelihood/cross-entropy loss function, but neural models are the most relevant example in this thesis.

of the same lexeme—a phenomenon that is far more visible in other languages where a single lexeme may result in hundreds of inflections. We will come back to this issue in Part II and Part III, but it also motivates the next chapter.

Chapter 3 (Mielke et al., 2021) is all about the old and fundamental problem of tokenization, or the question of "what are the fundamental units of language (processing)." After a historical review of the problem under the lens of linguistic plausibility we quickly establish the benefits of typographically defined pre-tokenization routines that broadly speaking split at spaces, perhaps splitting off punctuation, while mentioning the issue of reversibility that will be of relevance in the sequel of this thesis. Going further, we ask whether such simple word-like units perhaps are workable as long as models also have access to the underlying characters in a word (indeed, a version of this will be our tractable model introduced in Chapter 6). These models lead us to models that use character information to actually detect and decide on boundaries of word-like tokens, through multilevel RNNs, marginalization over paths, and Bayesian nonparametrics. Forgetting about these often complicated models, we then look at a simpler solution for finding units smaller than words intelligently to be open-vocabulary without having to go down all the way to characters: subword units, which are the by far most popular approach in the field today. We will look at manually constructed linguistically motivated units, those discovered by linguistically motivated algorithms, and finally units decided quickly and efficiently by essentially simple cooccurence statistics of the data, the most popular example for that last category being the omnipresent Byte-Pair Encoding (BPE; Sennrich et al., 2016). Becoming even more fine-grained, we take a look at what is often promised as the future: character- or byte-level processing. While the basic idea is easier to understand than more complicated open-vocabulary approaches, models based on characters or bytes suffer from a great many problems still not only with efficiency and while some of them may be remedied by going even further and decomposing into rendered pixels, we finish this chapter with a short discussion on why tokenization is likely here to stay as a topic in NLP and deserves more

thought and care than is often thought these days.

Chapter 4 finishes Part I by establishing the pre-tokenizers and baseline open-vocabulary language models used in this thesis. For the former, most space will be taken up by the description of a novel pre-tokenizer that emphasizes simplicity of implementation, reversibility (to a degree), and language-agnosticness, which we use for most preprocessing throughout the thesis. In terms of models, we define an open-vocabulary variant of a pre-neural n-gram language model for use in the pilot study of Chapter 7 and then devote the remainder of the chapter to *neural* open-vocabulary language models. Crucially, we will not only introduce RNN- and Transformer-based modeling skeletons, but also highlight how for the purposes of our thesis the underlying skeleton does not matter as much as a few key points of an interface of sorts: a hidden state or query that is compared against token embeddings to produce a distribution for the next token, and an autoregressive recurrence fed with the selected token's embedding. This means that although this thesis for the most part uses RNN-based models that some may consider outdated or at least out of fashion, the applicability of our ideas is not per se tied to RNNs, as we will make specific throughout. With this confidence, we define the neural baselines we will return to throughout the thesis: a character-level RNN language model and a BPE-subword-unit-level RNN language model, both of which are naturally open-vocabulary. We close the chapter with an often forgotten warning about segmentation ambiguity in BPE-based generative models and explain why this is not a problem for our work.

1.2 Part II: Building

In Part II, our objective, broadly speaking, is to advance language model theory past the stuck-in-2016 state that much of modeling finds itself in today when it comes to questions of vocabulary and tokenization. Consider the word "Dogecoin"[2] that we might see in

[2]A cryptocurrency that was started as a joke in 2013, but at some point actually became a relevant player in the cryptocurrency space.

financial reports. Neither the pre-2016 approach of replacing rare and unknown words with UNK nor the post-2016 segmenting of words into characters or subword units using simple heuristics like BPE and SentencePiece can give us an embedding of that word that is useful for downstream tasks: the former is unsuitable because it completely erases all information about the word in question, but the latter too is unsuitable as it might[3] break our example into "`Do@@ g@@ eco@@ in`." Using that segmentation we only get embeddings for uninterpretable units like "`g@@`" (and worse, units like "`eco@@`" that are ironically misleading given that Proof-of-Work-based technologies like Dogecoin are anything but `eco`-friendly) in place of a searchable, interpretable, and nicely composable unit. This places the burden of piecing together these meaningless subword units into the actual entities of interest on downstream models (or have humans deal with these undesirable results), going against linguistic knowledge and economical modeling. Motivated among others by such cases, we will first propose a grand model that is based on linguistic and modeling desiderata. While it is theoretically sound, we struggle to provide more than a sketch for an inference idea in this thesis (Chapter 5). To see if we are on the right track, we develop a feasible simplification that still satisfies most of our desired properties (Chapter 6).

Chapter 5 starts out with a simple idea: what if we could truly model a sequence of *latent* lexemes, automatically discovered during modeling, that are realized into spelled out words (or other useful units of any granularity). The crucial novelty is the idea of letting this inventory of lexemes be *infinite* in size, leading to the name of this "Infinite Neural Language Model" (relating to previous Bayesian nonparametric models that we essentially seek to "neuralize") and allowing the "discovery" of, say, Dogecoin even at runtime without requiring retraining. We posit our model as a generative process, in which we first generate the lexicon through a Pitman-Yor Process (Pitman and Yor, 1997) drawing from a prior distribution over embeddings as well as a neurally modeled spelling

[3]This is a real example, but of course given the flexibility of vocabulary size and implementation of BPE, it is not the only possible outcome.

distribution that generates character sequences given such an embedding, leading to a lexicon of infinitely many triplets of prior probability, embedding, and spelling. Generating text using this lexicon proceeds as usual, except now our softmax over these embeddings and associated lexemes is infinite in size—but importantly, not ill-defined, as multiplying in the "prior" probabilities of lexemes like a kind of unigram backoff results in a normalizable distribution! Only now, with the model description in hand, are we able to nicely introduce the linguistic background (duality of patterning and the arbitrariness of the sign) as well as the modeling considerations (type/token distinction and novel words) that this model adheres to. Of course, from here on things get difficult: how can we implement inference in this model—or even just generation from this infinite softmax? We will sketch an RJMCMC-based idea for inference and discuss various challenges before concluding that our time may be better spent building a model that compromises on the idea of latent lexemes but still respects the linguistic and modeling considerations we outlined. Before moving on to the next chapter that will do just that, we also revisit related work from Part I and show how our model is related to and improves upon selected papers.

Chapter 6 (Mielke and Eisner, 2019) defines a simplified model that has a finite vocabulary and no latentness in underlying lexemes, making this base form not too different from an ordinary closed-vocabulary neural language model with clever (i.e., in line with our linguistic and modeling considerations) regularization. To obtain an open-vocabulary model we first continue to make use of the classic UNK trick where we generate this UNK placeholder in place of a novel word. Then, however, we follow through on this promise by expanding each UNK into a fully formed spelling using the spelling distribution from our lexicon generation. As input to condition this distribution (which, again, is defined to be conditioned on a word embedding), we feed in the hidden state or query of the model at the location of the UNK token, allowing us to sample a hopefully fitting spelling for the UNK. While the model is still not trivial to implement, the fact that information flow between both levels is unidirectional during generation means that the model can be

trained extremely efficiently, at little extra cost over the closed-vocabulary model. After detailing this training procedure and the exact models used (including an interesting use of the nuclear norm in spelling distribution regularization), we move on to experimental validation. Experimenting on an English Wikipedia corpus, we show the superiority of our approach over our baselines and prior work, while admitting that on a multilingual corpus results are more mixed.[4] Quantitative analysis sheds light on performance in different word frequency bins; ablations and testing out different training and regularization strategies shows that all parts of our model do indeed perform a useful function. We also look at interesting qualitative results, not just in full model generations, but also in querying just the spelling distribution with embeddings of already existing (and learned) words to see what samples from this distribution look like. This is an interesting setup, as we would like these resampled spellings to be fitting in the contexts that the original words appeared in and thus be similar to the original spelling, yet we do not want them to look exactly the same, as this would mean overfitting on the largely arbitrary spelling of the original word. Indeed, we will find this is largely the case, encouraging us to use this model as a base for the more complex models of Chapter 5 in future work. We again finish this chapter by comparing this specific model to its related work and then close Part II out with a conclusion.

1.3 Part III: Evaluating

Part III then takes a step back and hones in on language model evaluation across more languages than just English. Specifically, we ask the question of how such models deal with the wide variety of languages of the world—are they struggling with some languages? And if so, is there some regularity to this struggle? We could relate this question to a more linguistic one, asking whether some languages are inherently more difficult to deal with

[4] It is in part for this reason that we will forego using it for the later massive multiligual experiments in Part III and stick to the already more established baseline models readers may find themselves more curious about.

(for a neural language model).

Chapter 7 (Cotterell et al., 2018) establishes the main ideas used here. The first important ingredient for fairly comparing performance is to ensure that all models are trained and tested on the "same" underlying data, just in their respective language—the perfect application for multi-text corpora like Europarl (Koehn, 2005) that pair any given sentence or paragraph with its translation in one or more languages. The second ingredient is to compare a metric that is not language-specific. We observe that perplexity and even classic open-vocabulary measures like bits per character (bpc) as a measure very much depend on the definition of a word or really a token in any given language, unfairly favoring some over others. We solve this issue by using the log-likelihood (or cross-entropy, surprisal, information content—it turns out there are many ways to say essentially the same thing here) of the entire translation unit. To make numbers live in the same intuitive realm as bpc, we arbitrarily divide all languages' bits by the number of characters in the English text,[5] giving us bits per English character (bpec) as a measure that is not only language-independent but also intuitively comparable to the bpc of a character-level (or any other open-vocabulary) model. Training both an open-vocabulary n-gram baseline language model and a character-level RNN language model, we see that indeed there are noticeable differences in performance and at least in this pilot study it seems like more complex inflectional morphology may correlate with such difficulty.

Chapter 8 (Mielke et al., 2019) tries to design a bigger and more sensitive and sensible study to drill into this claim and see if factors other than morphology are at play. The central novelty of this study is the carefully designed mixed effects model (for which we propose and analyze several variants), relating cross-linguistic content complexity of a sentence and a difficulty factor for each language to the surprisal of that sentence under a language-specific language model. We also not only drop the n-gram baseline in favor of a

[5]This may sound biased at first, but it essentially just means we choose an arbitrary constant that we divide all numbers by, so the ranking and relations do not change.

BPE-based RNN language model to compare with characters, but also analyze the effects of BPE hyperparameters across languages. On the topic of data, we expand too, using tens of different bibles (in what turns out to be an interesting optimization problem itself, requiring ILP solving), which allows us to look at more languages and language families, but also compare *different* Bible translations in the *same* language! Likely as a result of upgrading models to more realistic baselines, when we now again ask if there are good correlations relating a language's difficulty with its linguistic properties (like morphology) and more data-driven statistics (like the length of the test set in characters), we find that the claim of inflectional morphology driving differences is not tenable, and the only statistically significant correlations we find are with simple metrics of the raw data. We find solace in this surprising disappointment in the fact that our mixed effects model's ability to deal with missing data (to some extent) allows us to perform a comparison that before had been impossible: we can compare native text in a language with "Translationese," i.e., text written in that language as a translation of text originally written in another. With a careful setup, we can produce strong evidence against the common wisdom that Translationese is somehow easier to model than native text: our language models find both equally hard!

Chapter 9 (Bugliarello et al., 2020) closes Part III on this topic of translation, showing that the methods we have established for monolingual language models can be of great use in analyzing bilingual translation models as well, provided they are probabilistic generative models as well. We look at the case of neural machine translation in which probabilities (in other words, surprisals) for translations are easy to obtain. Once again, we prefer a surprisal-based metric over BLEU (Papineni et al., 2002), which is also highly language-and tokenization-dependent, showing that this allows us to compare translation difficulty from English into various languages (which is impossible with BLEU). But with all we established in the preceding two chapters, we can also go a step further and try to *separate out* the "monolingual generation" aspect of the translation process, essentially subtracting the surprisal of a monolingual language model on a given target sentence from the translation

model surprisal to get a measure of how much the translation model is struggling with the translation (or rather extraction and transformation) aspect of the task. We again look at typology and metrics of the data to find correlations (which again would be impossible with traditional metrics like BLEU), but once more find that raw metrics of the data seem most salient.

We close the thesis with a conclusion, summarizing our contributions and findings and pointing out some interesting directions for future work. Language model research does not need to revolve around data and compute moats and most impressive generation demos. There are and remain many questions and opportunities on both construction and evaluation as well as creative use of open-vocabulary language models.

1.4 Relevance in a rapidly changing field

It is true that in 2023, with huge models boasting billions of parameters the recurring topics of interest in this thesis—word formation, spelling, and segmentation—simply may not matter as much as they used to. Our findings and suggestions may not really apply to LLMs, as even without clever modeling, with enough data and perhaps some auxiliary tasks relating to spelling, an LLM may do just fine on the kinds of problems we consider.

That however does not mean that it is not worth understanding paradigms that are outside the limelight. For one, ideas and insights may transfer (and indeed we find much of what we show here to be applicable to, say, Transformers as well). But also, aside from general-purpose LLMs trained on predominantly English data, we must not forget that language models have been and will be used in more specialized contexts as well, where such resources, both in training data and in compute may simply not be available. And even if we were only interested in LLMs—their successful application rests on tireless efforts for efficiency to lower costs and increase availability, where clever modeling and thoughtful tokenization can quickly become hugely relevant.

Besides, isn't it odd that on these fronts we seem to have come up with no new ideas since BPE in 2016? Is that really the best we can do? Let us see what we can do to shake things up a bit.

1.5 Contributions

This finally leads us to a summary of the contributions of this thesis:

- a fairly comprehensive survey of tokenization efforts past and present illuminating background and connections (Chapter 3),
- a pre-tokenizer that emphasizes simplicity of implementation, reversibility (to a degree), and language-agnosticness (Section 4.1.2, released at `https://sjmielke.com/papers/tokenize/`),
- a unifying view on RNNs and Transformers for the purpose of neural autoregressive language modeling (Section 4.2.2),
- a theoretical open-vocabulary neural language model that is based on linguistic and modeling desiderata that are not met in existing models—only sketching inference ideas for the case of an infinite vocabulary of latent lexemes using Bayesian nonparametrics, as full-scale inference is out of scope for this thesis (Chapter 5),
- a practical open-vocabulary neural language model that is based on the same desiderata, but by giving up on infinite latent lexemes and instead using a finite vocabulary and a sort of back-off, is much more feasible to implement and train (Chapter 6),
- a BPE-RNNLM baseline that while common now, beat existing state-of-the-art models and sometimes even our own models at the time of publication (Sections 4.2.3 and 6.4),

- an experimental framework for fairly comparing language models' performance across languages and tokenization schemes, validated on 21 European languages (Chapter 7),
- a family of more sophisticated mixed effects models to calculate and compare language difficulty for given language models that allows for missing data at test time (Section 8.2),
- experiments using this missing data functionality to produce new insights in the relative difficulty of Translationese, showing it is different, but no easier to model (Section 8.6),
- experiments using the best-fitting model of this family to compare language models' difficulties with a total of 69 languages, evaluating different RNNLM types with different granularities in characters or different BPE settings, and different Bible translations in the same language (Section 8.4),
- a study investigating correlation between difficulty and linguistic as well as data-driven properties of languages, finding that the only significant correlations are with simple metrics of the data (Section 8.5), and
- a conceptual expansion of this probability-based performance comparison to bilingual translation models, showing that we can separate out the monolingual generation aspect of translation and compare bilingual translation difficulty across languages (Chapter 9).

Chapter 2

Background: Language Modeling

Published research and text

Parts of Section 2.5 in this chapter are taken from Mielke and Eisner (2019), parts of Section 2.6 from Wolf-Sonkin et al. (2018).

In essence, language models tell us what kind of text is likely to occur in terms of probability. Their history goes back further than that of the terms NLP and AI, not just through Claude Shannon's well-known "guessing game" to estimate the entropy of and efficiently encode English (Shannon, 1948, 1951) by means of character- and word-level n-gram language models, but all the way back to 1913, when Andrey Markov counted letters and n-grams in "Eugene Onegin" under the umbrella of his work on the thus named Markov Chains, resulting in the first probabilistic and generative model of running text (English translation: Markov, 2006).

As statistical NLP emerged in the late 1980s, language modeling became a central task in NLP, with the first models built for practical applications again being n-gram models. Already in the mid-1970s, Jelinek (1976) used such a model for speech recognition as part of a **noisy channel** setup in which the language model would effectively re-weigh the output of an acoustic model that would produce sequence candidates, formally specified by means of Bayes' rule, positing a generative story in which the language model generated a string and the acoustic signal transformed it into the observed signal, necessitating inference over the string given the signal.

Such re-weighting and re-ranking uses have been commonplace since, but especially in the past ten years, language models powered by neural networks have become a central ingredient of many NLP models themselves. The first sense in which that is true is that the decoders of generative models like those used for machine translation are implemented as neural language models over sentences in a target language that condition on a source sentence (Sutskever et al., 2014; Bahdanau et al., 2015)—and indeed it is from the use case of machine translation that NLP has embraced subword units like BPE for language modeling (Sennrich et al., 2016, see Section 3.5), which we will later examine.

The second way in which this is true is that language models have in the past five years been used as *feature extractors* (Howard and Ruder, 2018; Peters et al., 2018): since language models can be learned from nothing but raw text, it is easy to obtain lots of training data and build large neural language models whose internal representations, now called *contextual word embeddings*, are useful in downstream tasks, where they replace static word embeddings like word2vec (Mikolov et al., 2013a) for great gains; while ELMo (Peters et al., 2018) first popularized the idea, more recent application can make use of models like the OpenAI GPT family (Radford et al., 2018, 2019; Brown et al., 2020) family and the cloze-LM BERT (Devlin et al., 2018) and its successors.

Finally, since the release of GPT-2 (Radford et al., 2019), language models have even been practically used as text generators themselves, conditioned only on their own history, with the GPT-3 model (Brown et al., 2020) being the most recent and most impressive example of this. But we are getting ahead of ourselves. Let's start at the beginning.

2.1 What is language modeling, fundamentally?

Formally, a (generative) *language model* (LM) is a probability distribution over natural language sentences. Most commonly, such distributions are specified *autoregressively*, i.e., predicting one part of the structured space at a time given all previous predictions. The

structured space here is the space of sentences (or even entire texts, but for the conceptual introduction we will restrict ourselves to sentences) and the individual parts are *tokens*, a *sequence* of which makes up any given sentence. Traditionally tokens vaguely correspond to "words" or word-like units (Chapter 3 will go into great detail on this), splitting a sentence like "The cat sat." into four tokens: "The," "cat," "sat," and "." (treating the period as a token, too).

To define a probability, i.e., make a prediction for a sentence, we decompose the sentence's probability into individual probabilities or predictions left-to-right autoregressively:

$$\begin{aligned} p(\text{The cat sat.}) = p(\text{The}) \\ \cdot\ p(\text{cat} \mid \text{The}) \\ \cdot\ p(\text{sat} \mid \text{The cat}) \\ \cdot\ p(. \mid \text{The cat sat}) \\ \cdot\ p(\textsc{eos} \mid \text{The cat sat.}), \end{aligned}$$

where EOS denotes the end of the sequence and thus of prediction and the individual distributions are now over much a simpler unstructured discrete space: the set of all tokens known by the model, called the *vocabulary*, which we will denote by $\mathcal{V}$. For the time being let us simply assume that this is a finite set as it has been in many models. This is obviously at odds with reality, but that conflict warrants its own discussion in Section 2.3.

Armed with this autoregressive idea of decomposing the structured probability of a sentence into tokens, the main question in building language models[1] is: how do we define the individual token predictions? As sentences go on, the *context* or *history*, i.e., the preceding words that we condition on, grows longer and longer, making it harder and harder to define these distributions.

[1]Language models of the autoregressive left-to-right variety, that is; other ideas for non-autoregressive modeling as well as different directions and orders of tokens for the decomposition have been studied but are a small niche compared to the models that are widely used.

2.2 From n-grams to LSTMs

The traditional language models of the last century, mentioned in the opening of this chapter, have solved this dilemma through the introduction of a conditional independence assumption that essentially posits that only recent tokens matter for prediction. Termed *n-gram models*, the main parameter n governs the size of the window relevant for prediction, namely that only the last $n-1$ words matter. Simplifying $p(. \mid \text{The cat sat})$ into, say, a 3-gram model yields a simpler probability $p(. \mid \text{cat sat})$ that ignores earlier words like "The," deeming the two preceding words enough for prediction. That (admittedly somewhat crude) simplification allows the model to be specified as a table of probabilities whose size is cubic in the size of the *vocabulary*, i.e., the set of all possible tokens (more on this later). Training a model like that essentially boiled down to counting n-grams like "cat sat ." in a large corpus, normalizing the counts to obtain probabilities, and in practice smoothing these counts to deal with sparse signals, which becomes more crucial as the size of the context n grows.

Despite much research on this smoothing process, context sizes have generally remained less than 10 tokens even with very large amounts of data. One reason for this is of course that at that point n-grams get so specific that they only appear once leading the model to overfit on only the specific sentences found in the training data unless extensive back-off or other forms of regularization are used. The other reason, however, is that the number of parameters in a traditional n-gram model with probability tables grows *exponentially* in n (limited only by corpus size) and so larger context size models become intractable to train or even just store.

Wouldn't it be nice if we could somehow have a way of modeling that kept the numbers of parameters small by *sharing* information between n-grams? In a way older tricks like hashing made some progress on this front, but the real breakthroughs came with the introduction of *neural networks* into language modeling. Bengio et al. (2001) essentially

kept the n-gram structure but probabilities now came from a neural network instead of a table of probabilities as before. This neural network would take in a sequence of $n-1$ tokens, encoded as finite-dimensional real-valued vectors, termed *embeddings* (leading to the known term *word embeddings* when tokens roughly correspond to words), concatenated into a single vector of still-fixed size (due to the fixed value of n) that could then be run through a multilayer perceptron to be transformed into a *hidden state* vector. This vector would have the same dimensionality as the word embeddings and by taking dot products with all embeddings in the vocabulary and exponentiating and normalizing (in short referred to as *softmax*) we would obtain a probability distribution for the next token.

The neural sharing of parameters helped a great deal with the aforementioned issues, but *neural language models* would need another innovation to become a household name. In 2010 this innovation was put together by Mikolov et al. (2010), who removed the simplifying assumption of finite and fixed-size context that is inherent to all n-gram models by employing *recurrent neural networks*. In these, the idea of a hidden state encoding all of the context to use in prediction returns, but this time around, it is built up step by step, in particular token by token. At the beginning of a sentence, the hidden state is initialized to a fixed vector, and then for each token in the sentence, a new hidden state is calculated from the previous one and the current token's embedding by means of a simple feed-forward neural network (usually just a simple matrix multiplication and nonlinearity). This way, we never need to consider just any fixed context size n, as through this evolution of the hidden state, tokens that are arbitrarily far back still inform the current prediction. The limit to this is of course the size, i.e., the dimensionality, of the hidden state that can realistically only carry so much information to be used by later prediction steps, but crucially, that limit is somewhat soft and can be raised by increasing the size of the hidden state.

The next somewhat obvious improvement is the introduction of multiple *layers* in the RNN itself turning the *hidden state* into a tuple of individual layers' hidden states which are passed on not only to subsequent layers but also to the current layer in the next timestep.

The expressive power of such models is greatly increased, but we are not quite done yet for there is one last important innovation that is needed to make the neural language models we consider in this thesis: *long short-term memory* (LSTM; Hochreiter and Schmidhuber, 1997) cells. This and other variations of RNNs (like gated recurrent units (GRU); Cho et al., 2014) augment the hidden states of our layers with an equal-size *cell state* that is manipulated only through addition and subtraction and gives rise to the original hidden state component that is used for prediction itself, making it easier to retain information for longer amounts of time, truly fulfilling the promise of long context sensitivity.

LSTM language models are the end of our whirlwind sketch of the evolution of language models here, but we will give them a proper formal introduction in Section 4.2.2, where we will also give mention to the Transformer (Vaswani et al., 2017) architecture that in 2023 has effectively replaced RNNs for language modeling and other NLP tasks but whose inner workings are not necessary to understand the contributions of this thesis and in particular the following sections.

2.3 What makes a language model open-vocabulary?

When we introduced the idea of a vocabulary $\mathcal{V}$ for which we maintain embeddings in neural models and over which the individual predictions at each time step range, we said to assume that this set was finite. However, assuming that the tokens are essentially words, the way language changes and training data always being limited in some way we run into problems when trying to assign a probability to or sample words we have never seen before.[2] Should these have 0 probability? That would mean the entire sequence does, which clearly is not desirable. Such words are termed out-of-vocabulary (OOV) and in this section we will see what happens when you essentially ignore them and hint at solutions to actually model these words, too.

[2]Often 5–10% of held-out word tokens in English language modeling datasets were never seen in training data. Rates of 20–30% or more can be encountered if the model was trained on out-of-domain data.

The easiest and historically dominant approach to avoid catastrophic undefinedness in the face of OOVs has been to first define a vocabulary $\mathcal{V}$ containing only sufficiently frequent words and then simply replace all words that are not in $\mathcal{V}$ with a distinguished symbol UNK (*unknown* word)—both in training data and when the model is actually used. The resulting class of models is called *closed-vocabulary* and while the solution works in that it results in a well-specified model, a model like that will not be able to distinguish between different OOVs in sentences and will not be able to generate them at all. This works well enough for re-ranking outputs of other systems that are unlikely to produce OOVs to begin with, but in 2023, this approach is clearly insufficient for a number of reasons:

1. For natural language generation (NLG), UNKs in the output are not acceptable.

2. For feature extraction using large-scale models like ELMo (Peters et al., 2018) or BERT (Devlin et al., 2018), novel words often are very useful anchors of meaning and not just one-off events (Church, 2000).

3. For use in languages other than English, in particular those with more productive morphology, UNKing makes effective modeling hard if not impossible, as we will see in Part III of this thesis when we will show how it also prevents comparisons between languages.

What is needed in this age of language modeling are *open-vocabulary* language models, i.e., models that can generate novel words when needed—but that means anticipating an *infinite* set of words for each individual prediction...

Or does it? Once we let go of the idea that the tokens that are autoregressively predicted are words, an easy solution presents itself: we can simply predict *characters* instead of words, or even bytes in any encoding. Those clearly are very much a finite and in fact quite small set, so nothing will ever catch us unaware! While this works, it means that a *lot* more steps are necessary to predict a given sentence, making models slower, and on top of that harder to train as the lengths of dependencies between meaningful elements (say, subject

and verb) of a sentence also increase greatly in terms of computation steps. When we talk about tokenization in Chapter 3, we will look more closely at tradeoffs between words and characters as well as the "compromise" of *subword units* like BPE (Sennrich et al., 2016) that finds much application in practice.

2.4 Evaluating language models

There are many different ways of evaluating language models, which makes sense considering the wide variety of tasks they are used for. Such *extrinsic* evaluations are always preferable when a specific task is already known, but for a researcher trying to build better language models, fast, easy, and task-independent metrics for *intrinsic* evaluation of a model are highly desirable. Luckily the probabilistic formulation of language modeling lends itself to clean probablistic evaluation on held-out text data.

Given a language model, i.e., a probability distribution, p and some test sentence s, we can obtain a sentence's probability $p(s)$ but we can also transform this number into one with a slightly different interpretation: the **surprisal**, calculated as the negative log-likelihood $\text{NLL}(s) = -\log_2 p(s)$. This surprisal can be interpreted as the number of bits[3] needed to represent the sentence under a compression scheme that is derived from the language model, with high-probability sentences requiring the fewest bits.

Long or unusual sentences tend to have low probabilities, i.e., high surprisals—but high surprisal can also reflect a language's model's *failure to anticipate* sentences that really *should* be predictable. The latter view leads us to the core of all subsequent metrics: the overall probability/surprisal of an entire test corpus. This number is convenient because it doesn't depend on tokenization or model granularity (character-level vs. word-level vs. subword-level of different granularities) since we just ask about the probability of the entire test corpus, not individual tokens. We easily obtain the total probability by

[3]This is obviously only the case when indeed using $\log_2$ specifically. ML frameworks often just use log, so the quantity is expressed in "nats."

multiplying individual probabilities of tokens in all predictions, or likewise, we obtain the total surprisal by summing over the surprisals of all tokens in the corpus.

The downside behind a global number like this is that it tends to be fairly unwieldy and imposing and that it is very clearly dependent on test set size. Total surprisal in particular scales linearly with the size of the test corpus, meaning a larger test corpus would give us a larger surprisal, looking "harder" to model. To make results across test sets more comparable (even though of course every test set is different so numbers are never exactly comparable unless that difference is the point) and to make numbers more human-readable, we often go back to token-level metrics: **perplexity** for closed-vocabulary models and **bits per character** (bpc) for open-vocabulary models.

Of these two, bits per character is very intuitive given our explanation of surprisal above and can be trivially obtained by dividing total surprisal (i.e., total bits) by the total number of characters in the test corpus. In the case of character-level models, this works out to be equivalent to the average surprisal at each prediction step in the test corpus, but as we will see all throughout the remainder of the thesis, not all open-vocabulary models are character-level, in which case the detour through total surprisal is necessary.

Perplexity on the other hand is often somewhat of a mystifying concept to new students of NLP as it seems to be a number neither in the well-understood realms of probabilities or of bits. It is defined as the exponentiated surprisal or bits per *token* $e^{-\log p(s)/|s|}$, where the number of tokens $|s|$ very much depends on the tokenization and model class used, as usually it is calculated by summing up surprisals over each prediction step and dividing by these prediction steps before exponentiating (with the same base as used as in the logarithm) the entire thing.

Intuitively, perplexity behaves like a reciprocal probability. This becomes clearer if we rewrite it as $e^{-\log p(s)/|s|} = \sqrt[|s|]{e^{-\log p(s)}} = \sqrt[|s|]{1/p(s)} = 1/\sqrt[|s|]{p(s)}$, i.e., the geometric mean of the reciprocal probabilities or the reciprocal of the geometric mean of individual probabilities. This reciprocal lends itself to a nice interpretation: perplexity tells us how many wrong

guesses the model had on the average step before it hit the right token to predict, which makes obvious that lower is better.

The firm dependency on the number of prediction steps and thus tokenization that is left unspecified in the bare concept "perplexity" wasn't much of an issue in the early days of NLP, as word-level models were the de facto standard and minor tokenization differences, when present and not standardized, didn't change results all that much. However in the modern age, where even if you call subword-level NLP a de facto standard that still doesn't tell us what granularity of subword division is used, this metric becomes completely useless for comparisons across models.

Or does it? In Mielke (2019) we give a slower explanation of the aforementioned concepts to make explicit that as long as you know the granularity, or, really, the total number of tokens used in the calculation of whatever perplexity metric you encounter, you can work your way back to obtaining the total probability or surprisal and from that calculate perplexity in your own chosen tokenization scheme, essentially converting perplexities easily between tokenizations.

The result of conversion being possible is that there is no need to rewrite your favorite language modeling toolkit that only spits out bpc or perplexity as long as you know the total number of tokens in whatever units you need for the test set.

We will use these ideas somewhat matter-of-fact without much mention in Chapter 6, but they will be at the core of our thought process in Chapter 7 and continue to be used in the remainder of Part III.

For completeness' sake, we will also mention the concept of **cross-entropy**, which conceptually measures a kind of "distance" between two distributions (not a distance in the mathematical sense, as it is not symmetric, in fact, it is not even a divergence as it is not 0 when the two distributions are equal), here the true test distribution (i.e., the test corpus) and the model distribution. Its definition of $H(p, q) = -\mathbb{E}_p[\log q]$ for two distributions p

and q breaks down into a simple average in the case of finite discrete test corpora where p defines only point masses, the test examples $x \in X$: $H(p, q) = -\sum_{x \in X} p(x) \log q(x)$. From here on we encounter a similar question of granularity: is our test set one long example containing a sequence of sentences or are the sentences independent? This independence very much changes the resulting number as it changes the very way the model is run over text in the first place! As independence of sentences has become broken with the advent of neural language models that can keep long contexts, this metric has faded into the background in modern language modeling, as when truly the entire test set is a single example (i.e., the corpus is just one long string), the data distribution assigns all its mass on that one example and cross-entropy simplifies to $-\log q(x)$, which is simply the surprisal or negative log-likelihood of the test example under the model, nomenclature that makes clearer what is going on. We will, however, come back to the terms of entropy and cross-entropy at the end of the thesis in Chapter 9 when considering machine translation, where independent sentences are still a common assumption.

Finally, to close out this section, we should note that there are many evaluations somewhere between extrinsic and intrinsic evaluations that zoom in on specific aspects of generation under some decoding strategy or the assignment of probability before decoding that don't depend on a specific target task but also do look at specific phenomena or use cases. Examples are strands of work that hone in on accuracy of syntactic agreement, e.g., Linzen et al. (2016a), gender bias (Lu et al. (2020), expanded for morphologically rich languages in our work not present in this thesis, Zmigrod et al. (2019)), and accuracy or quality of statements expressed in the output of generation, evaluated either by humans or by automatically comparing to references (a realm in which many modern LLM evaluations live, e.g., Wang et al. (2019a), but also a little ways from the beaten path of standard evaluations work of our own on calibration in LLM-driven chatbots not in this thesis, Mielke et al. (2022)).

2.5 How to Train your Language Model

In this section we will cover the training of neural sequence models, specifically the RNNs we use in this thesis, though Transformers share everything mentioned here. The general idea for almost all neural model training follows the idea of *hill-climbing* on an **objective function** that measures how well we fit the training data. This means that we start with randomly initialized parameters (drawn from some specific distribution) and then update these parameters step by step, each step hopefully improving the objective function. In language modeling, the objective function usually implements the idea of *maximum likelihood* (ML), by *minimizing* the negative log-likelihood (NLL) of the *training* data, which as mentioned above is equivalent to minimizing the cross-entropy between the training data and the model distribution.

As hinted at above, we will regard the training data, too, as one long tokenized sequence, with each line break (as a stand-in for a sentence boundary, sometimes resulting in the generation of entire paragraphs) replaced by an end-of-sentence EOS token (thus, $\text{EOS} \in \mathcal{V}$). Thus deviating from the standard exposition that equates EOS with the end of prediction, prediction both at training and test time goes on until a length limit is reached. This idea of just a single long (potentially infinite) sequence is standard practice in the language modeling community at least since Mikolov et al. (2010), as it allows the model to learn long-range content-sensitive dependencies, i.e., relationship that cross sentence or line boundaries.

The actual maximization of probability makes use of neural network architectures usually being differentiable through **backpropagation** by using the **gradients** of the parameters with respect to the training NLL, updating parameters iteratively using **stochastic gradient descent** (SGD; Robbins and Monro, 1951), in each step subtracting the gradients to move the parameters in the direction of a lower NLL. Stochastic Gradient Descent implies that our gradients are only estimates of the true full gradient—the reason

for that is that it is impractical to compute the gradient of the language model likelihood on a single string of millions of tokens. We thus usually perform two simplifications to make training feasible.

First, let us look at the length of the sequence. The process of unrolling the computation of any recurrent model into one big graph so backpropagation proceeds both down layers of the model and back through time is aptly called **backpropagation-through-time** and it is standard practice in language modeling[4] to only obtain gradients for a *short sequence* at a time, e.g., a single sentence or a string of fixed or stochastically sampled length. Such short sequences are usually non-overlapping across the corpus, but may include the previous context when using RNNs, perhaps even through stochastic choice as in Merity et al. (2017a). The process of using short substrings not only greatly reduces memory requirements (making the process feasible to begin with), but also allows for more frequent updates on partial data gradients, hence the stochasticity in the gradients.

Second, instead of training just on one single sequence at a time and having to deal with high-variance gradients, a number of sequences of equal length are combined into a **minibatch** or **batch** that can be run through the model in parallel on modern accelerator architectures with results in the case of gradients being added up or averaged over strings in the batch. This greatly increases processing speed while also reducing the variance of individual gradients, which now are sums or averages of a number of individual gradients. Notably, this still retains an unbiased estimator of the overall gradient, so the overall procedure still falls cleanly under the SGD paradigm.

Finally, each such gradient step usually also includes **regularization** like the **weight decay** operation that shrinks all parameters of the model toward 0, probabilistically corresponding to a Bayesian interpretation of having a zero-centered Gaussian prior on the parameters and performing **maximum a posteriori** (MAP) inference rather than pure

[4]As noted by Mikolov et al. (2010) and reflected even in reference implementations of frameworks like PyTorch: https://github.com/pytorch/examples/tree/0.3/word_language_model

maximum likelihood.

2.6 Inflectional morphology

Let's close out this chapter with something that will come up in various parts of this thesis: *inflectional* morphology.

The majority of the world's languages overtly encodes *syntactic* information *on the word form itself*, a phenomenon termed **inflectional morphology** (Dryer et al., 2005). In English, for example, the verbal lexeme with **lemma**[5] "talk" has four inflected forms: talk, talks, talked and talking. In a language like Archi, however, Kibrik (2017) calculates a theoretical possibility for more than a *million* verbal forms, formed by combining **morphemes** of different axes of syntactic meaning with the lemma.

Given our discussion of finite vocabularies and OOV words in Section 2.3, it is clear that inflectional morphology poses a challenge for language modeling: if we want to model Archi, we need to be able to generate all of its inflected forms, but we can't possibly store all of them in our vocabulary. This is one of the reasons why open-vocabulary modeling becomes a downright necessity in morphologically rich languages. If we stick to a finite set of *words* in our model, so much of text will be hidden as OOV and metrics will be meaningless as well as generation being unusable.

This is why we believe it important to evaluate open-vocabulary language models not just on English, where there is little morphology at play, but also on languages where there's plenty of morphology—and that is what we will do not only in Chapter 6 but also far more expansively in Chapters 7 and 8. In the latter two, we will further attempt to quantify the degree of morphological inflection using **morphological counting complexity** (MCC; Sagot, 2013). This crude metric counts the number of inflectional categories distinguished by a language (e.g., English includes a category of 3rd-person singular present-tense verbs),

[5]The concept of a lemma is not the same as the more NLP-centric one of a word stem, but for the purposes of this thesis, we won't need to go into great detail on this here.

as annotated in the language's UniMorph (Kirov et al., 2018) lexicon.

Chapter 3

Background: Tokenization

Published research and text

This chapter is largely taken from Mielke et al. (2021).

> *"'tokens' are not a real thing. they are a computer generated illusion created by a clever engineer"* —@dril_gpt[1]

As we have seen in the previous chapter, questions of modeling language, especially with concerns about novel words and intra-word processing are fundamentally influenced by the question of what the units of text are that we want to base our models on. To give this question the treatment it deserves, let us go back to the very start.

When we first introduce people to NLP models, we often take for granted the idea that text is cut up into little pieces that are fed to a computer, eventually as nothing but a sequence of integers. Following Webster and Kit (1992), we call these (usually) contiguous substrings *tokens*. In teaching settings and in fact historically in NLP, these tokens are somewhat naturally implied to be *words*—at first, perhaps naively as "space-separated substrings" in English. Understandably, as soon as we try to split punctuation from words, things get a little tricky. Take the word *"don't"* for example: not splitting it is reasonable,

[1] A Twitter bot with (human-curated) outputs of a language model based on GPT2 (Radford et al., 2019) and trained on tweets of Twitter poet @dril; https://twitter.com/dril_gpt2/status/1373596260612067333. While the original post most likely referred to the cryptocurrency scam adjacent "non fungible tokens," it fits the tone of this chapter well: tokenization is an engineering *choice*.

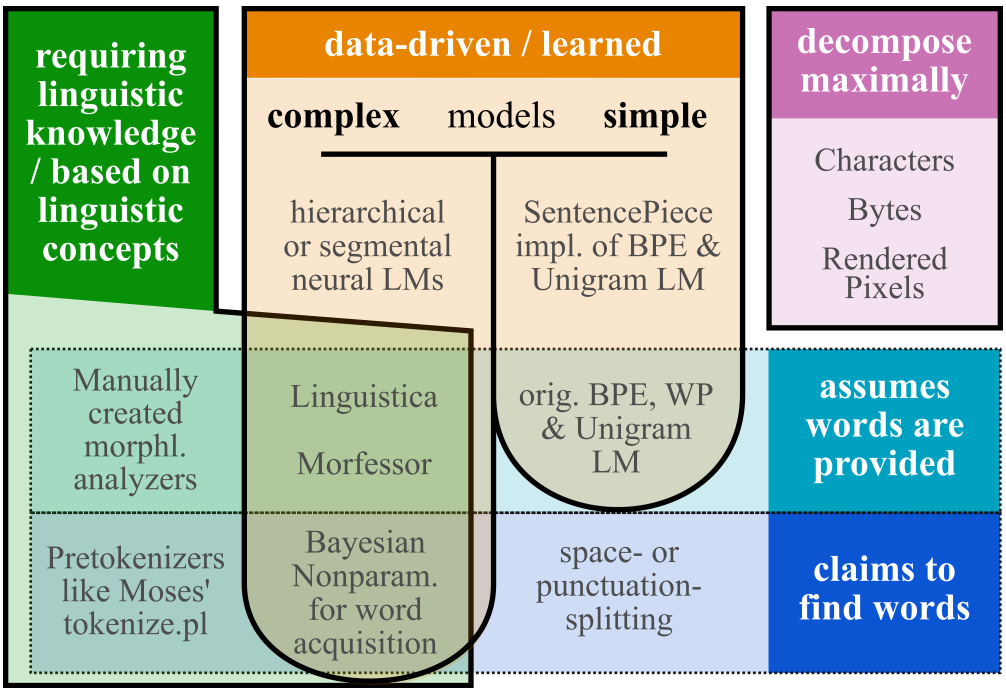


Figure 3-1. A taxonomy of segmentation and tokenization algorithms and research directions

but if we split on all punctuation, we would get three somewhat nonsensical tokens (*don ' t*)—and we might argue that the most sensible split actually ought to yield the two units "*do*" and "*n't*", as found in the Penn Treebank (Marcus et al., 1993).

We will elaborate on fundamental questions and terminology of *tokenization* in Section 3.1, and show how this important if somewhat unglamorous part of all NLP work has historically been treated (Section 3.2). However, since especially in the last five years there has been renewed interest in going *beyond* intuitive definitions of a "token" as a somewhat atomic word-like space-separated unit. One way to do so is to use word-internal information to augment word-like units (Section 3.3), which neural modeling has made easier than ever, leading to models that can even *learn* word boundaries with no overt indication (e.g., when spaces are not present). That notion of unsupervised word segmentation or discovery has been present on its own in decades of work (Section 3.4), but we will find it useful to consider when finally looking at the now prevalent idea of using subword units as atomic tokens (Section 3.5). We will finish our descent with work using the simplest tokenization

possible: maximal decomposition into characters, bytes, or even pixels (Section 3.6).

Equipped with all this knowledge, in Section 3.7, we will be making a case for why "complicated" tokenization is something to practice or even learn about even now in the 2020s when the easy solution of bytes seems in reach. We will argue that while recent advances like CANINE (Clark et al., 2021), ByT5 (Xue et al., 2021), or Charformer (Tay et al., 2021) make maximally decomposed processing feasible for certain domains and use-cases, they do not cover the wide variety of NLP scenarios and come with their own drawbacks and biases.

3.1 Tokens, word-forms, and sub-words

In NLP, textual data has been traditionally segmented into "sentences" (or "utterances", etc.) and "words" due to linguistic motivations and technical constraints. The macroscopic units ("sentences") are often considered independently from one another and themselves segmented into microscopic units. The definition of these microscopic units has always been a matter of approximation and compromise. On the one hand, these units receive linguistic annotations (e.g. part-of-speech tags, morphosyntactic annotation, syntactic dependency information), which would require them to be linguistically motivated units. On the other hand, a large range of phenomena make it highly non-trivial to identify and even to consistently define linguistic units, denoted by the Morphological Annotation Framework (MAF) ISO standard (Clément et al., 2005) as *word-forms*. Such phenomena include contractions (e.g. English *don't*), compounds (e.g. German *Entzugserscheinungen* 'withdrawal symptoms'), morphological derivatives (e.g. English *anti-mask*), as well as numerous classes of named entities and other sequences following type-specific grammars (e.g. numbers, URLs).

As a result, *typographic units* have been used as an approximation for such linguistically motivated units and denoted *tokens*. For instance, MAF defines a token as a "non-empty

contiguous sequence of graphemes or phonemes in a document." In the case of writing systems using a typographic separator such as whitespace, now universally used with the Latin script for instance, *tokens* have been widely used and broadly defined as either punctuation marks or contiguous sequences of non-punctuation non-whitespace marks. As long as a few arbitrary decisions are made regarding certain punctuation marks (e.g. the hyphen or the apostrophe), a definition like that makes it possible to deterministically split a sentence into atomic units, resulting in a segmentation into tokens that are acceptable approximations of word-forms. Crucially, as discussed in detail by Clément et al. (2005), Sagot and Boullier (2008), and elsewhere, there is no one-to-one correspondence between tokens and word-forms; a word-form can be made of several tokens (e.g. English *hot dog*) whereas several word-forms can be represented by the same token (e.g. English *don't* = *do* + *not*). This is what the Universal Dependencies guidelines[2] refer to as "multitoken words" and "multiword tokens," respectively, a topic further discussed by More et al. (2018). In fact, both phenomena can interfere in nontrivial ways (e.g. French *à l'instar du* = *à_l'instar_de* + *le*).

It is worth noting that when the writing system at hand does not have a typographic separator, tokens must be defined differently. With scripts like the Chinese or Japanese script, an option for instance is to consider each character as a token on its own.

In recent years, the spread of approaches based on neural language models resulted in an evolution in how sentences are split into atomic units, thereby resulting in a redefinition of the notion of tokenization. Indeed, based both on scientific results (e.g. the impact of sub-word segmentation on machine translation performance (Sennrich et al., 2016)) and on technical requirements (e.g. language models that want to be open-vocabulary yet require a fixed-size vocabulary), the need for the atomic processing units (still called tokens) to be an approximation of word-forms has faded out. As a result, in current NLP, the notion of token still perfectly matches its MAF definition, but it no longer corresponds to the traditional

[2] https://universaldependencies.org/u/overview/tokenization.html

definition of a typographic unit. "Tokenization" now denotes the task of segmenting a sentence into such non-typographically (and indeed non-linguistically) motivated units, which are often smaller than classical tokens and word-forms, and therefore often called *sub-words*, as we will see in Section 3.5. Typographic units (the "old" tokens) are now often called "pre-tokens," and what used to be called "tokenization" is therefore called nowadays "pre-tokenization." This term is motivated by the fact that the first approaches to the new notion of "tokenization" often involved segmenting sentences into proper typographic units (i.e. the "old" notion of tokenization) first, before further segmenting (some of) the resulting units (formerly "tokens", now "pre-tokens") into eventual "sub-words." We will therefore be calling such approaches "pre-tokenization" from here on, anticipating the later processing into subword units.

3.2 Pre-tokenization yields word-like typographic units

Many tools, formerly known as "tokenizers" and nowadays as "pre-tokenizers", have been developed and used for a long time. Some of them are relatively simple and remain faithful to the typographic token. Amongst the most widely used, we can cite the venerable Moses (Koehn et al., 2007) tokenizer[3] and the more recent `pre-tokenizers` package in Hugging Face's Tokenizers package.[4] Their use saw "tokenization" (now "pre-tokenization") reduced to the task of segmenting sentences into atomic units in general, i.e "basic units which need not be decomposed in a subsequent processing" (Webster and Kit, 1992), even when such units are closer to word-forms than to typographic units.

It is fairly common for tokenizers to not only segment sentences but also modify the raw text, for instance for *normalization*, spelling correction or named entity detection purposes, thereby departing from the standard definition of token. Thus a string like "some 'quoted' text" might be tokenized into five units: "some " quoted " text" (note the normalization of

[3] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl

[4] https://github.com/huggingface/tokenizers

apostrophes). Normalization operations like these or conflation and merging of different whitespace symbols leads to most tokenizers being *irreversible*, i.e., we cannot recover the raw text definitively from the tokenized output.[5,6]

3.3 Augmenting word-level pretokenizer tokens with character information

As explained in Chapter 2, word-level models are conceptually easy to understand and in the neural era (Bengio et al., 2001; Collobert and Weston, 2008; Mikolov et al., 2013b) offer features at an interpretable granularity, but their central weakness is the inability to deal with rare and novel words, i.e., words that were seen very rarely during training or not even at all (out-of-vocabulary, OOV)—they are *closed-vocabulary* models. Nevertheless, under the assumption that the word *is* a fundamental unit of language, a number of approaches emerged to improve handling of rare and novel words under a fundamentally word-based framework by basing their handling on the characters that make up a word.

3.3.1 Augmenting word-level models with spelling information

The idea of somehow using information about the spellings of a word to inform the word's representations of course is decades old. In neural models for language, research in the 90s and 2000s often forewent the focus on words altogether and processed strings of characters instead (see Section 3.6 and Section 3.3.2), but as soon as neural models became important in NLP, combinations of word- and character-level information for use in neural networks emerged there, too.

[5]While this is not a major issue for most applications, it means that we no longer model the original text, but a string that may correspond to many original strings, inflating probability estimates of language models; this issue is also highlighted in the context of ambiguous tokenization (see Section 3.5.1.3) by Cao and Rimell (2021).

[6]The "reversible" language-agnostic tokenizer of Mielke and Eisner (2019), described in Section 4.1.2, attempts to remedy some of these issues, but still conflates different kinds of whitespace.

Dos Santos and Zadrozny (2014) first proposed to use information about the spellings of words themselves to aid word embedding estimation. Soon thereafter, Ling et al. (2015), Kim et al. (2016), and Jozefowicz et al. (2016) popularized the idea of deterministically constructing a word's embedding from its spelling,[7] both for textual input as well as for generative language modeling, that is, prediction of strings. However, even when replacing learned word embeddings with those constructed from a word's characters with convolutional neural network (CNN) layers, their generative models are still *closed-vocabulary*, meaning they can only predict words that were seen (often enough) in training data, so the CNN construction only helps with rare words, not novel words. Furthermore, constructing embeddings from spellings for each token (as opposed to each type like we will do in Part II) implicitly trains the CNN-powered embedding function to "get frequent words right" instead of anticipating rare or even novel words, an issue discussed further in Section 3.3.2. Similar constructions led to advances in other classic NLP tasks like POS tagging (Plank et al., 2016) and ultimately powered the first big *contextual word embedding* model, ELMo (Peters et al., 2018).

The popular fastText embeddings (Bojanowski et al., 2017) propose constructing word embeddings not from characters, but from overlapping n-grams, allowing one to obtain embeddings for novel words just like we could from underlying characters (making it "open-vocabulary" in that sense, though not in the generative sense). Ataman and Federico (2018) likewise obtain better performance on machine translation by using (overlapping) n-grams instead of characters (also beating BPE on morphologically rich languages).

In more recent times, El Boukkouri et al. (2020, CharacterBERT) and Ma et al. (2020, CharBERT) use the same CNN construction as in Kim et al. (2016) on a modern BERT-style model, this time enhancing the BPE units' embedding with their constituent characters' embedding, motivated by better handling noisy texts with spelling errors or transferring

[7] It should be noted that Jozefowicz et al. (2016) also propose a variant in which output tokens are not scored through a softmax, but generated character by character, anticipating the advancements described in Section 3.3.2, but still staying in a closed-vocabulary setup.

to new domains like medical text; concurrently, Aguilar et al. (2021) do almost the same, but using a small Transformer instead of CNNs.

Finally, construction-based approaches have also been integrated into pretrained word-level input models. Specifically, Pinter et al. (2017) learn a model that is trained to *mimic* the embedding of a word given its spelling using a helper RNN model that is called whenever an unknown word appears during test time. Without going into too much detail yet, we will end up turning this idea on its head in Part II.

3.3.2 Open-vocabulary language modeling with (tokenizer-defined) words made of characters

Extending closed-vocabulary generative models to open-vocabulary models, i.e., those that can predict and generate novel words at test time, is somewhat more difficult than being open-vocabulary on the *input* side because it must be possible to hold out probability mass for the infinite set of sentences that contain completely novel words.

Inspired by Luong and Manning (2016), we will propose a probabilistic two-stage model in Chapter 6 that essentially augments the ordinary closed-vocab word-level recurrent neural network language model (RNNLM) setup by regularizing word embeddings to be predictive of their spellings using a smaller character-level RNNLM and using that smaller model to generate novel words on the fly whenever the word-level RNNLM predicts UNK, yielding an open-vocabulary model motivated by linguistic notions and intuitive modeling and proven successful qualitatively and quantitatively.

Independently developed, the model of Kawakami et al. (2017) follows a similar two-level setup of word- and character-level RNN, but where each word has to be spelled out using a character-level RNN if it cannot be directly *copied* from the recent past using a cache model (Grave et al., 2016).[8] Their analysis shows clearly that the cache model not

[8]As mentioned before, the idea of spelling out words in isolation from hidden states had previously proven unsuccessful in Jozefowicz et al. (2016)'s comparison, but this was in a closed-vocab setup and without the caching mechanism Kawakami et al. (2017) employ.

only copies "bursty" unknown words like `Noriega` (Church, 2000), but also extremely common function words like `the` in an attempt to keep itself from forgetting them. The same kind of two-level setup is reinvented by Ataman et al. (2019) for a machine translation decoder (creating word embeddings on the encoder side from character-level BiRNNs as in ELMo (Peters et al., 2018, see Section 3.3.1)) and later extended by Ataman et al. (2020) with some additional stochasticity that is intended to pick up on lemmata and inflections unsupervisedly.

A different approach is having higher layers of multi-layer RNNs run at lower speed (skipping updates to the hidden state) This is an old idea, first present in El Hihi and Bengio (1995) (building on Schmidhuber (1991, 1992)'s "neural sequence chunker") and revived in Koutnik et al. (2014) for fixed-frequency skipping and Hwang and Sung (2017) for skipping between word boundaries (which are assumed to be observed).[9] This approach leads to the first of a number of ways in which we can actually *learn* word boundaries and thus segmentations.

3.4 Learning segmentations to find concatenative word-like pretokenizer tokens

So far we have relied on having a predefined notion of word (or pre-token) despite the conceptual struggles outlined in Section 3.1. But what if such a definition is not given, not obtainable, or simply not desirable (for reasons of robustness and in languages other than English etc.)? Is there a way to let our data-driven machine learning approach also *learn* the tokenization? Most approaches described in this section propose to tackle tokenization by treating the implied *segmentation* as a latent variable (with an exponentially-sized domain) on which we can perform approximate or (using more assumptions) exact inference to find

[9]Specifically, Hwang and Sung (2017) describe an architecture in which character-level and word-level models run in parallel from left to right and send vector-valued messages to each other. The word model sends its hidden state to the character model, which generates the next word, one character at a time, and then sends its hidden state back to update the state of the word model.

segments and boundaries that hopefully correspond to meaningful units. The various techniques described in this section yield units of varying size and quality.

3.4.1 Character-level neural models that learn to skip steps at higher levels

Already in the 1990's, Elman (1990) manually analyzed character-level RNNs and correlated their prediction surprisal with word boundaries. This idea that was then expanded on in Schmidhuber (1991, 1992)'s "neural sequence chunker". More recently, surprisal was applied to not only character-level neural models but also n-gram models under a beam search framework by Doval and Gómez-Rodríguez (2019) to split microblog texts in which spaces are deleted.

Instead of using post-hoc surprisal thresholding, the HM-RNN (Chung et al., 2017) takes the idea of multiple timescales motivated in Section 3.3.2, but *learns* the binary decision to skip or update (thereby providing a sense of word boundaries), optimizing with approximate gradient descent using the straight-through estimator (Bengio et al., 2013). In their model, communication between layers happens bidirectionally: the lower network reports its final state to the higher one; that higher network reports its new state to the lower layer that then proceeds to run by itself and so on. While they "recover" word boundaries when including spaces in their data, Kawakami et al. (2019) claim to get unusable segments with this model when not including spaces. Furthermore, when trying to use the HM-RNN for NMT, Cherry et al. (2018) report that it took a lot of fixing to get it to train at all; its performance on the task was competitive but not superior. This finding corroborates that of Kádár et al. (2018), who dedicate a paper to trying to get the HM-RNN to train well, ablating it, and also showing subpar segmentations on text data (as well as the worrying inability to reach the original reported numbers). Kreutzer and Sokolov (2018) try to use a similar paradigm of skipping steps and generating summaries with lower layers for NMT and find (similarly to Kádár et al. (2018)) that skipping is rarely

used and thus seems to be unnecessary for good performance. Nevertheless, the model is extended to phrase- and sentence-level boundaries by Luo and Zhu (2021).

It is worth pointing out that despite having coarser layers of computation, these models still have to "spell out" a word every time it is generated, i.e., they cannot *memoize* tokens as reusable units.

3.4.2 Marginalization over all possible segmentations

Finally, a conceptually straightforward approach is to treat the segmentation of a string as a latent variable that needs to be marginalized over both at training and test time. This essentially means having a vocabulary that contains strings of differing lengths that overlap, i.e., it may contain "cat," "at," and "foster cat," such that the string "my foster cat" can be decomposed a number of ways corresponding to different sequences of latent units. As the number of segmentations is exponential in the sequence or context length, we need to either resort to approximations for marginalizing over latent decompositions (Section 3.4.2.1) or simplify the model with independence assumptions e.g. by using an n-gram model (Section 3.4.2.2).

3.4.2.1 Approximate marginalization

Chan et al. (2017) propose an estimator to approximate the marginal probability of observations using approximate MAP inference through beam search. They find that the model is very hard to train, but manage to obtain promising results. Buckman and Neubig (2018) confirm this estimator's instability and propose some approximate inference schemes based on averaging RNN hidden states that produce better results in terms of LM perplexity. Hiraoka et al. (2020) implement a similar model, based on a Unigram LM tokenization proposal distribution (see Section 3.5.1.3), whose n-best tokenizations of a sentence are fed into any sentence encoder model independently and whose resulting sentence embeddings are averaged in line with their a priori tokenization likelihood. Hiraoka et al. (2021) extend

this model to sequence-to-sequence settings by training a tokenizer and downstream model with separate losses, the former by rewarding tokenizations that produced a low downstream loss, and the latter using just one tokenization sampled from the conditioned (and tempered) LM.

3.4.2.2 Exact marginalization using additional independence assumptions: segmental neural language models

The more popular solution of *segmental neural language models* was pioneered by Kong et al. (2016), who cast the problem of segmentation as a kind of monotonic[10] seq2seq task, going from a sequence of characters to a covering sequence of substrings, i.e., a segmentation, performing a kind of BiRNN-CRF chunking. This works by conditioning segment prediction on the *entire raw string*, processed and embedded using a BiRNN. Segmentation decisions/scores can thus use context, but by scoring every individual possible substring independently as a segment using these embeddings and then adding up individual scores to score entire segmentations, they can find a covering of the entire input string with segments efficiently using dynamic programming. The reason for this ability is the central independence assumption: the model does not depend on any *other segments* when scoring a segment, but merely on surrounding *characters*. Wang et al. (2017) extend this by also having a per-segment RNN over characters for the outputs that can run without knowing the segmentation and whose past representations can thus be used by the individual segment generation processes, allowing for left-to-right sharing of information about segments without breaking dynamic programming.

The jump to LMing is now made simply by omitting the conditioning on an input (and with it both BiRNN and CRF), yielding the model of Sun and Deng (2018), who coin the term segmental language model, training on Chinese characters and using the unsupervisedly learned segments to compete on Chinese Word Segmentation. To keep

[10]Interestingly, with some reordering of the input one can break monotonicity between input and output, making the model similar to phrase-based MT (Huang et al., 2018).

the computation of the quadratic number of segments feasible, they restrict the segments to a maximum length of 4 characters (a sensible prior for Chinese). Grave et al. (2019) make the same jump independently, using Transformers as the independent character-level global backbone. When evaluating on English open-vocabulary language modeling, Grave et al. (2019) notice improved perplexity, but not using or evaluating the obtained segmentation, most likely because they, too, only use 4-grams that appear at least 400 times. Contemporaneously, Kawakami et al. (2019) use the same independence idea, but have emissions of string segments come from a context-dependent *mixture* of a character-level model like in Kawakami et al. (2017) (see Section 3.3.2) and a large set of substrings (up to 10-grams that appear often enough in training data) with learned embeddings. They evaluate not only on perplexity, but also on word segmentation performance, where they do beat some baselines (see Section 3.4.3), but still perform much worse than some previous models,[11] which they argue tuned their hyperparameters on segmentation performance instead of marginal likelihood and thus have an unfair advantage.

Interestingly, when training on image captions, Kawakami et al. (2019) find that both perplexity and segmentation performance improve when the model also has access to the image that is being described, showing that learning segmentation only from monolingual and unimodal text may be harder than when other modalities or languages are present. This observation is shared by He et al. (2020), who build a similar segmental model (in their case, a Transformer-based version that still follows the character backbone idea to allow for dynamic programming) as the target-side generator in an NMT system and use it not as the final model, but merely as a learned tokenizer. This is easy to achieve by changing the dynamic program from marginalization to maximization and thus obtaining a new segmentation, called DPE, that can be used in place of BPE or unigram LM (see Section 3.5). He et al. (2020) proceed to show that learning to tokenize with a small

[11] On English they cite the pre-neural models of Johnson and Goldwater (2009) and Berg-Kirkpatrick et al. (2010) as significantly better; on Chinese, they are beaten by pre-neural models like the one of Mochihashi et al. (2009) and the neural model of Sun and Deng (2018). More information about some pre-neural models is given in Section 3.4.3.

Transformer-based NMT model[12] produces better segmentations than BPE for use in a bigger model; in particular, training the tokenizing model on the translation task produces different segmentations depending on the source language, and, more importantly, better segmentations (as measured through downstream translation performance) than training on target-side language modeling alone.

The idea of conditioning on characters and predicting segments is extended to the (non-directional) masked language modeling setting found in Transformers and left-to-right autoregressive Transformers by Downey et al. (2021), though results do not outperform RNN-based SNLMs consistently.

Note that many of these models can also be seen as relatives of models based on UnigramLM, which we will cover in Section 3.5.1.3.

3.4.3 Finding words through Bayesian non-parametrics

In the era of n-gram and word-based language models, MacKay and Peto (1995) noticed that a Bayesian view of autoregressive language models may prove beneficial, reinterpreting smoothing and backoff in n-gram models as inference in a hierarchical model where higher-order distributions are drawn from a Dirichlet distribution whose mean is a lower-order distributions.

Teh (2006) builds on this idea to propose a *hierarchical* Pitman-Yor Process (PYP; Pitman and Yor, 1997) language model where we again have n-gram distributions of arbitarily large orders (though still finite), drawn through a hierarchy of PYP distributions that lead to a model that still bears resemblance to n-gram language model smoothing. The pinnacle of this idea of modeling was reached in Wood et al. (2011)'s sequence memoizer, which removed the requirement for a finite n, boasting great compression performance for arbitrary binary data and still performing very well on language modeling tasks, although

[12]Unlike previously mentioned papers, they however restrict the vocabulary to units of an input BPE vocabulary instead of using length and frequency heuristics.

neural models at this time already proved to be strong competitors.

Focusing not on infinite context but more on word formation, Goldwater et al. (2006b) used a Bayesian perspective to tackle an infinite vocabulary, explaining how new words are first coined and how they are then used in running text: a process they call two-stage language modeling (see Section 3.3.2), with the two stages being referred to as *generator* (which creates new lexemes) and *adaptor* (which governs reuse; here, a PYP), relating the resulting interaction between types and tokens to interpolated Kneser-Ney smoothing as presented in Chen and Goodman (1999).[13]

Given such a two-stage model to explain text and the use of Bayesian nonparametrics that can assign positive probability to an infinite number of possible lexemes, it becomes possible to also try to infer word boundaries, that is to perform unsupervised word segmentation. Motivated more by trying to explain and model cognitive processes and in particular child language acquisition, Goldwater et al. (2009)[14] summarize Unigram and Bigram Dirichlet Processes (DPs) for segmenting transcribed infant-directed speech, showing superiority over older non-Bayesian approaches. Mochihashi et al. (2009) extend the idea from bigram DPs to ∞-gram nested/hierarchical PYPs to improve word segmentation for English written text; Elsner et al. (2013) additionally model phonotactic processes that convert a sequence of segments into observed realizations.

3.5 Learning subword vocabularies and segmentations

As teased in Section 3.2, *subword* units allow for a smooth transition between a word-level model and a character-level model: split the word-like tokens obtained by pre-tokenization into smaller units: the set of all possible subword units is finite and can be determined from training data, but it is assumed to include all characters (or bytes, see Section 3.6.3)

[13]The formalism of generators and adaptors is extended and formally specified under the name *adaptor grammars* in Johnson et al. (2007) and used very successfully for state-of-the-art word segmentation in Johnson and Goldwater (2009).

[14]The idea and partial results are already presented in Goldwater et al. (2006a), but the authors request citing the updated 2009 paper. Goldwater et al. (2011) summarized this thread of research.

that appear at test time, making it possible to explain any novel word in held-out data.

While thinking about subword information may have more tradition for processing morphologically rich languages, Mikolov et al. (2012) already proposed using *subword units*[15] instead of words for language modeling English to avoid out-of-vocabulary (*OOV*) issues. Only since Sennrich et al. (2016), however, has it become customary for NLP models to use subwords to avoid large or infinite vocabulary size.

It is important to point out that despite reasonable motivation, segmentation may be a bad idea in for example Semitic languages like Arabic and Hebrew (Shapiro and Duh, 2018) or other languages with non-concatenative morphological phenomena, which Amrhein and Sennrich (2021) claim are better served by character-level models or those with very small subword inventories.

What then should we use as subword units, or, in other words, where does this inventory come from? One option is manually constructed, linguistically informed rule-based systems, another is given by data-driven segmentation learners, which traditionally have been motivated and evaluated either linguistically (e.g., Morfessor: Creutz and Lagus, 2002) or given by simple heuristics to be fast and easy and improve downstream performance. We will focus on the latter in this section, information about the former and references to various comparisons between them evaluating their performance can be found in Mielke et al. (2021).

3.5.1 Algorithm choices

3.5.1.1 BPE (Gage, 1994; Sennrich et al., 2016)

BPE is a compression algorithm from a family termed "macro-schemas" (Storer and Szymanski, 1982) in which substrings are replaced with references to them (containing among others the famous Ziv-Lempel algorithms like Ziv and Lempel (1978)). The name was coined in Gage (1994), although equivalent algorithms have been applied for pattern

[15]They don't supply many details, but claim that the units they use are *syllables*—and that they help.

discovery in natural language (Wolff, 1975) and complexity of genetic sequences (Ángel Jiménez-Montaño, 1984) earlier.[16] When learning a tokenization, BPE replaces pairs of adjacent symbols with a new symbol representing that pair, iteratively merging all occurrences of the pair that occurs most often at any given time. At test time, the same procedure of merging can be performed by executing all recorded merges in the order in which they were conducted during training of the tokenization model.

Byte-Level BPE (Wang et al., 2019b) applies BPE not to characters, but raw bytes (see Section 3.6.3); it is used in GPT-2 (Radford et al., 2019) and other models. BPE-dropout (Provilkov et al., 2020) is an extension allowing for subword regularization (see Section 3.5.1.3).

3.5.1.2 WordPiece (Schuster and Nakajima, 2012)

A very similar technique had been proposed under the name "WordPiece" by Schuster and Nakajima (2012) for Japanese and Korean text (where reliance on space-separated tokens is impossible as text is written without spaces), though it is also used in BERT (Devlin et al., 2018) and other models. Unlike BPE, WordPiece doesn't merge the most often co-occuring pair but pairs that increase the likelihood that an n-gram based language model trained with this updated vocabulary reaches on data (the fact that only some counts need to be updated in such a model and the use of frequency-based heuristics for selecting and batching of multiple merge candidates keep the process computationally feasible). To segment text, WordPiece follows a per-word left-to-right longest-match-first strategy, allowing for very fast linear-time processing (Song et al., 2021).

3.5.1.3 UnigramLM (Kudo, 2018)

Kudo (2018) picks up on the idea of judging subword candidates by evaluating their use in a language model, but it uses a simple unigram language model (hence calling the algorithm

[16] See Gallé (2019) for more historical connection and corresponding analyses, e.g., its linear-time implementation by Larsson and Moffat (2000).

unigram LM) and iteratively *removes* subword units from a starting vocabulary that contains far more subword units than are desired: on every iteration, the unigram LM is trained using EM and then the lowest-probability items are pruned from the vocabulary—the process is repeated a few times until a desired vocabulary size has been reached.

Interestingly, this probabilistic setup also cleanly models the fact that there are many possible segmentations that are consistent with a given string (in the extreme case, one could always fall back to characters). They report that training with *sampled* segmentation (termed "subword regularization") instead of just using one deterministic segmentation indeed improves machine translation performance. The same motivation led Provilkov et al. (2020) to propose BPE-dropout where the skipping of individual merges in the BPE construction leads to variety in segmentations. Subword regularization not only has been shown to help in monolingual in-domain tasks, but also for improving transfer in multilingual models, see additional discussion in Mielke et al. (2021).

The observation that sampling segmentation helps is confirmed by Hiraoka et al. (2019), who employ a Bayesian nonparametric model (see Section 3.4.3) as the LM that defines the tokenization.

Wang et al. (2021) build a similar model where the unigram LM is based on character-level BiLSTM encodings of the input and apply it to unsupervised Chinese Word Segmentation.[17]

3.5.1.4 SentencePiece (Kudo and Richardson, 2018)

Not itself an algorithm as often assumed, but actually a software package, SentencePiece (Kudo and Richardson, 2018) offers both BPE and Unigram LM algorithms (so specifying "SentencePiece" is certainly not informative enough). Importantly, unlike their other implementations it does not treat spaces as special guaranteed word boundaries, allowing

[17]Note that this thus character-conditioned model can also be seen as an example of segmental neural language models (Section 3.4.1).

learned units to cross these boundaries and obviating the need for pre-tokenization in languages without whitespace-tokenized words like Chinese and Japanese.

3.5.2 How many units do we need?

Another open question is the question of how many merges (or what other prior) one should select for optimal performance. The best-performing number may depend on the task and the domain and differ by language (Domingo et al., 2018; Mielke et al., 2019, i.e., Section 8.1.3).[18] More and thus larger subword units allow for and lead to more memorization (Kharitonov et al., 2021), which may or may not be desired depending on the application.

Predicting the number of merges that works best without having to try different sizes would be desirable and Gowda and May (2020) claim to have found one such heuristic: merge as much as possible to shorten the overall sequence lengths while making sure that 95% of subword units appear at least 100 times (presumably in training data). Their motivation is that neural machine translation models are frequency-biased in their outputs and thus maintaining a more uniform frequency distribution is better.[19] A similar study undertaken by Ding et al. (2019) reiterates how contradictory suggestions for number of merges in past work are and add that in low-resource scenarios far fewer merges than commonly expected seem to be better, a trend with Transformers which differs from that with LSTMs, leading to an interesting question: should smaller corpora mean you can't afford characters or is it rather that you can't afford words?

A simple online answer to the question of how to select merges is presented by Salesky et al. (2018): while training an NMT model using BPE segments, gradually *increase* the

[18]Relatedly, Novotný et al. (2021) show that the subword size matters and differs somewhat systematically between languages in the n-gram based fastText embeddings (Bojanowski et al., 2017).

[19]A somewhat similar approach is taken by Gutierrez-Vasques et al. (2021), who look at the entropy (and transformations) of the distribution over subword types to identify a "turning point" at which one of the transformed quantities is minimal—but this turning point happens with far fewer merges than are generally required to reach good performance.

vocabulary size by merging BPE vocabulary items, adding new, bigger BPE segments until they obtain diminishing returns for their downstream metric. Embeddings for the newly introduced subwords are initialized by merging the embeddings of the two merged BPE segments with an autoencoder. Formulating the vocabulary selection problem as a search for the set of tokens with the highest entropy, Xu et al. (2021) proposes an optimal transport driven selection from BPE units that obtains vocabulary merges that often outperform a language-independent standard setting for translation. Another recent method that comes with a stopping criteria (and therefore dispenses with an additional hyperparameter) is Vilar and Federico (2021) which defines the likelihood of a vocabulary with respect to a sequence, and improves that likelihood greedily.

3.6 "Tokenization-free" character- and byte-level modeling

In sections Section 3.3.1 and Section 3.3.2 we discussed augmenting word models with character-level information in closed- and open-vocabulary settings. The idea of pure character- or byte-level modeling seems like an obvious simplification. Indeed, Sutskever et al. (2011) successfully modeled strings one character at a time using multiplicative RNNs, Chrupała (2013) suggest using character-level RNN modeling for agglutinative languages and tasks that require character information like character-level text segmentation (even anticipating contextualized embeddings!), and Conneau et al. (2017) successfully perform text classification from raw characters. The big breakthrough for generative character/byte-level models however came only with Al-Rfou et al. (2019), who showed that sufficiently deep Transformers (64 layers in their case) can *greatly* outperform previous subword-based and hybrid (Section 3.3.2) open-vocabulary language models. This finding was updated by Choe et al. (2019), who again manage to match previous word- and subword-based state-of-the-art language modeling results.

3.6.1 Characters?

A major factor limiting the adoption of character-level models is the fact that character sequences tend to be much longer than their word- or subword-level counterparts, making training and inference slower. To improve training speed and efficiency, Libovický and Fraser (2020) propose to start with a subword-based model and then fine-tune that model to instead work over characters, though they find improvements only in one of two evaluated language pairs. The more common approach to both training and inference however are various architectures for subsampling such long sequences, particularly in applications to machine translation: Chung et al. (2016) introduce a bi-scale recurrent neural network to enable the decoder of an encoder-decoder model to produce character sequences; they demonstrate improved performance over a subword-level decoder. Lee et al. (2017) (and later Gao et al. (2020)) advocate for the use of convolution and pooling layers at the input of the encoder. Cherry et al. (2018) evaluate various temporal pooling strategies including the HM-RNN of Chung et al. (2017) (discussed in Section 3.4.1) and conclude that none of them offered a suitable trade-off of performance and speed.

It has been argued that character-level models are more robust to noise and out-of-distribution data (Gupta et al., 2019), possibly because a word- or subword-level token sequence will change dramatically in the presence of noise. Libovický et al. (2021) however survey multiple character-level MT systems and conclude that they "show neither better domain robustness, nor better morphological generalization, despite being often so motivated," a finding corroborated by Rosales Núñez et al. (2021) for noisy user-generated text. Specifically under the lens of gender bias, Gaido et al. (2021) argue that character-level processing can lead to less gender bias in models: data-driven BPE vocabularies are biased towards male forms, for one because of frequency, but also because in languages like French and Italian, female forms are often constructed by affixing to male forms, leading to a disparity in tokenization. They show that character-level processing can ameliorate these

disparities to some degree.

3.6.2 Character hashes?

In massively multilingual settings, naïve use of a character-level model can result in a very large vocabulary. For example, Unicode 13.0 specifies 143,859 codepoints, most of which will never be seen in training. This can lead to "out-of-vocabulary" problems just like the ones encountered with words (discussed in Section 3.3). One possible solution is used in Clark et al. (2021)'s CANINE, where instead of giving each character its own embedding, they *hash* all possible codepoints with multiple hash functions into smaller sets to share such embeddings across codepoints that are then concatenated across the multiple hash functions (similar to Svenstrup et al. (2017)). CANINE further includes downsampling and upsampling operations to attain reasonable computational efficiency, and focuses on non-generative sequence labeling tasks.

3.6.3 Bytes?

An alternative way to avoid the huge-vocabulary problem of character-level models is to instead represent text using the byte sequence resulting from a standard encoding like UTF-8. This can be seen as using a fixed tokenization scheme that was created by a standards body (the Unicode consortium) rather than a linguistically- or statistically-informed process. The main advantage of using Unicode bytes specifically is Unicode's incredible scope – the Unicode consortium states that their goal is to cover "all the characters for all the writing systems of the world, modern and ancient" in addition to "technical symbols, punctuations, and many other characters used in writing text".[20] Separately, byte-level modeling is presented as a natural choice by Graves (2013) when following the "principle of modelling the smallest meaningful units in the data". As shown recently by ByT5 (Xue et al., 2021), byte-level models confer similar benefits to character-level models (better robustness to

[20] https://unicode.org/faq/basic_q.html

noise, avoidance of out-of-vocabulary issues, etc.) with similar drawbacks (longer training and inference time due to longer sequences). As such, similar results and models have been proposed for byte-level models as for character-level. For example, the recent Charformer model (Tay et al., 2021) downsamples input sequences to avoid increased computational costs, and Al-Rfou et al. (2019) demonstrate strong language modeling performance with a deep byte-level Transformer.

3.6.4 So are these maximal decompositions the solution then?

While byte-level modeling is often presented as an unbiased, token-free approach, we argue it is more constructive to consider byte-level modeling as simply using a fixed, predefined, and standardized tokenization scheme (that incidentally is often the same as the way that underlying the text data is stored on disk). This tokenization scheme is by no means the "best" or most fundamental – indeed, the Unicode standard was not created with any linguistically-motivated goal in mind, apart from being able to represent a huge variety of content. Indeed, using a Unicode-based byte-level tokenization scheme will result in significantly different trade-offs for different languages by virtue of the way the standard was created: Latin (i.e. ASCII) characters are represented as a single byte, whereas characters in other languages are represented as multiple bytes. An immediate consequence is that a UTF-8-based byte-level tokenization scheme can result in dramatically longer sequences when representing the same underlying semantic content in different languages. This could unfairly increase computational costs for downstream users of a model who do not communicate in ASCII symbols.

3.6.5 Visual featurization: Pixels?

Another approach to "tokenization-free" modeling utilizes *visual* rather than byte-based representations. Where byte-level models aim to cover the full underlying 'vocabulary' as represented *on disk*, visual representations aim to encode similarities among characters

which *human readers* may use when processing text. One motivation is robustness: byte-level models are reliant on consistent observed sequences, which leaves them susceptible to variation and noise which may render similarly visually.[21]

The initial motivation for much work on using visual features were to create embeddings that reflected the shared character components found in e.g., Chinese characters (radicals) or Korean syllable blocks, and so were better able to generalize to rare or unseen characters. The first work in this space initialized embeddings for Chinese by linearizing bitmaps of characters or words (Aldón Mínguez et al., 2016; Costa-jussà et al., 2017). Subsequent work focused on character segmentation, either through pre-computed embeddings based on linearized images (Wang et al., 2020) or learned with the downstream task via convolutions (Liu et al., 2017a; Dai and Cai, 2017; Broscheit, 2018; Meng et al., 2019), improving results for rare characters. Character segmentation was in part motivated by application to Chinese and the focus on sub-character components, but also enabled the use of fixed-width images, a problem which Sun et al. (2018, 2019) instead tackled by rendering words into square images, wrapping characters as necessary.

Recent work has proposed *"tokenization-free"* visual representations (Mansimov et al., 2020; Salesky et al., 2021). Rather than rendering images for a given segmentation into words, characters, or bytes, and thus incorporating their previously discussed challenges, they render each sentence as an image and translate directly from pixel decompositions. Salesky et al. (2021) show that such models can perform competitively across a range of languages and scripts for machine translation and are significantly more robust to induced noise, including but not limited to unicode errors. Mansimov et al. (2020) explore pixel-to-pixel models; a challenging setting that is not yet competitive but neatly sidesteps many concerns with text-based models.

[21]For languages without a digital orthographic standard (e.g., Pashto), multiple byte sequences render similarly and are in free variation among speakers.

3.7 Discussion: is tokenization still relevant?

Tokenization, both the process and the term itself, has evolved significantly since the early days of NLP. In this chapter, we traced their evolution and highlighted major changes over the years, connecting disparate intuitions.

Despite significant advancement, we have seen that there is no perfect method of handling tokenization. All options—from whitespace-delimited pre-tokens to learned subwords and down to bytes—have drawbacks and strengths. Many desiderata may be fundamentally at odds with each other, e.g., the desire to decompose maximally for simple and robust processing with a desire to be computationally efficient in a way that is fair across languages—a question that particularly pertinent as the field turns its attention to greener NLP. While there are applications for which characters and bytes or even pixels may be the tool of choice, there are a myriad of applications in which larger discrete tokens are desirable, both for interpretability and efficiency. Recent work gives us new examples of this: Zhang et al. (2021) improve BERT by feeding inputs tokenized multiple ways and Dossou and Emezue (2021) use human-annotated larger-than-word vocabulary units to improve low-resource MT. Itzhak and Levy (2021) show that using subword units need not mean giving up on spelling-information, showing that spellings can be recovered from subword-based pretrained models.

In conclusion, despite all the promises of neural networks of allowing "end-to-endness", tokenization is a clear testimony that that promised land is still far away. Like often, this drawback is evident for practitioners who have to verify that the pre-trained models matches the tokenizer that is being used (a separated model in most NLP frameworks). The fact that tokenization is handled completely independent of the down-stream tasks adds to this separation. As Henderson (2020) puts it: "It remains to find effective neural architectures for learning the set of entities jointly with the rest of the neural model, and for generalising such methods from the level of character strings to higher levels of

representation".

Chapter 4

Pre-Tokenizers and Baseline Models for this Thesis

Published research and text

Parts of this chapter are taken from Mielke and Eisner (2019) and Cotterell et al. (2018).

Armed with all this knowledge about language modeling and questions of tokenization, especially as they pertain to open-vocabulary modeling, we can now outline the specific pre-tokenizers and models that we will use in this thesis.

4.1 Pre-Tokenizers

For the models in this thesis, we will always need pre-tokenizing for our data, no matter the eventual modeling strategy that subsumes the finer-grained tokenization choices. Given the different uses and the different times of publication of the original work, we will use both pre-existing pre-tokenizers as well as a novel one of our own.

4.1.1 Existing pretokenization schemes

Chapters 7 and 9 will rely on existing pre-tokenizers, in particular the tokenization script of Moses (Koehn et al., 2007) commonly used with the Europarl data used in both chapters, mentioned before in Section 3.2.

Again, choices like the use of BPE we will not cover in this subsection as they pertain

more to actual modeling choices in Section 4.2.

4.1.2 A novel simple, language-agnostic pre-tokenizer designed for reversibility

For use in all other subsequent chapters (Chapters 5, 6 and 8), we introduce a very simple pre-tokenization routine that relies on Unicode categories to not only split on spaces (as for example Kawakami et al. (2017) do) but also by splitting off each punctuation symbol as its own token. This allows us to use the same embedding for a word[1] regardless of whether it has adjacent punctuation. For fairness in comparison, we would like our pre-tokenizer to preserve all information from the original character sequence (i.e., reversibility), though we will only achieve this to a degree (detailed in the next paragraph). The exact procedure—which is simple and language-agnostic—is described in this subsection, with accompanying code.

4.1.2.1 Universal character categories

The Unicode standard defines all symbols in use in current computer systems. In it, each symbol is assigned to exactly one "General category", e.g., Lu for "Letter, Uppercase", Ll for "Letter, Lowercase", Sc for "Symbol, Currency", or Cc for "Other, Control".

We define the set of "*weird*" characters, i.e., characters we want to break the string on as those whose category does not start with L (i.e., letters), with M (i.e., marks like accents), or with N (i.e., numbers), and which are not "*space*" either, where "space" is defined as a character that Python's `str.isspace()` method returns true on.

It would be tempting to use Z*, i.e., the Unicode "Separator" category, as this third option, but since Python classifies some control characters (i.e., characters in Cc) as spaces, we use this behavior to ensure compatibility with Python whitespace splitting, which is the way pre-tokenizers are commonly included through (i.e., instead of a list of strings, the

[1]We will call numbers, i.e., digit sequences, words here, too.

pre-tokenizer returns a string where spaces separate pre-tokens that the modeling toolkit then splits using Python's `split()` method).

It is the whitespace-splitting in particular that makes our subsequent use of this tokenizer not entirely reversible, as spaces, tabs, and other whitespace are consumed as boundaries by the `split()` method in Python, and output is concatenated with spaces unless other information-preserving steps are taken.[2]

4.1.2.2 Tokenize

To tokenize a string, we look at each character c_i of the string (with i ranging from 1 to n):

1. If it is not *weird*, output it as it is.
2. If it is weird, we need to split and leave markers for detokenization, going through each of these steps:
   (a) If $i > 1$ and c_{i-1} is not *space* (i.e., we are really introducing a new split before this weird character), output a space and a merge symbol "⍇".
   (b) Output c_i.
   (c) If $i < n$ and c_{i+1} is not *space* (i.e., we are really introducing a new split after this weird character) **and** not *weird* (if it is, it will just split itself off from the left context, no need to split now), output a merge symbol "⍇" and a space.

Tokenization thus turns a string like "Some of 100,000 households (usually, a minority) ate breakfast (before 12)." into "Some of 100 ⍇,⍇ 000 households (⍇ usually ⍇, a minority ⍇) ate breakfast (⍇ before 12 ⍇) ⍇." using the Unicode character "⍇" (Leftwards Arrow To Bar Over Rightwards Arrow To Bar, U+21B9) that is assumed to not occur in underlying data.

[2] A possible extension of this would be to treat the ASCII space character as privileged and treat all whitespace that isn't that exact character, as well as subsequent space character after a first one as special punctuation subject to the same treatment as other punctuation. To stay similar to past tokenizers, we opt not to do this, but if that is an important concern, it should be trivial to define this extension.

Note that punctuation clusters are separated into individual tokens. Note also how merge symbols attach to punctuation rather than letter sequences in an attempt to aid generalization, allowing word tokens to appear in various punctuation configurations.

A consequence of this design is that there are sequences of token types generated by this method that are not the result of a valid tokenization run, e.g., "a ⸮,⸮ ⸮,⸮ a"—a string that would be decoded into "a,⸮,a" by the detokenizer detailed next, containing an undesired but not catastrophic merge symbol in the output. It is important to point out that using this tokenization for a neural generative model means that even if the model learns to generally not generate such sequences, there will be probability mass left for them that will result in subpar probabilistic evaluation results. We will revisit this kind of issue in Section 4.2.3.3, arguing its irrelevance for the remainder of this thesis.

4.1.2.3 Detokenize

Again, we look at each character c_i of the string that is to be detokenized:

1. If c_i is a space, c_{i+1} is the merge symbol "⸮", and c_{i+2} is *weird*, skip ahead to c_{i+2} (i.e., undo a right split).

2. Otherwise, if c_i is weird, c_{i+1} is the merge symbol "⸮", and c_{i+2} is a space, output c_i and move on to c_{i+3} (i.e., undo a left split).

3. Otherwise, just write out c_i and then continue to c_{i+1}.

4.1.2.4 Python implementation

In summary, the relevant methods are displayed in Python/pseudocode in Fig. 4-1. A full and commented implementation can be found at `https://sjmielke.com/papers/tokenize`.

```
MERGESYMBOL = '↹'


def is_weird(c):
  return not (unicodedata.category(c)[0] in ['L', 'M', 'N']
              or c.isspace())


def tokenize(instring):
  for i in range(len(instring)):
    c = instring[i]
    c_p = instring[i-1] if i > 0 else c
    c_n = instring[i+1] if i < len(instring) - 1 else c

    if not is_weird(c):
      stdout.write(c)
    else:
      if not c_p.isspace():
        stdout.write(' ' + MERGESYMBOL)
      stdout.write(c)
      if not c_n.isspace() and not is_weird(c_n):
        stdout.write(MERGESYMBOL + ' ')


def detokenize(instring):
  i = 0
  while i < len(instring):
    c = instring[i]
    c_p = instring[i-1] if i > 0 else c
    c_n = instring[i+1] if i < len(instring) - 1 else c
    c_pp = instring[i-2] if i > 1 else c
    c_nn = instring[i+2] if i < len(instring) - 2 else c

    if c + c_n == ' ' + MERGESYMBOL and is_weird(c_nn):
      i += 2
    elif is_weird(c) and c_n + c_nn == MERGESYMBOL + ' ':
      stdout.write(c)
      i += 3
    else:
      stdout.write(c)
      i += 1
```

Figure 4-1. Simplified version of our tokenizer in Python/pseudocode

4.2 Models

Given the sketch of what language models may look like from Chapter 2 and our knowledge of tokenization choices from Chapter 3, we can now outline the specific models that we will use in this thesis.

To specify models, we need the following basic notation.

Let Σ be a discrete alphabet of token types (be they words, subwords, characters, or something else), including a distinguished unknown-character symbol $\bigstar$.[3] The standard language model definition over finite strings ended by the end of sequence symbol EOS (which as already stated in Section 2.5 we will deviate from in practice) then defines the probability of a string $\mathbf{w} = w_1 \dots w_{|\mathbf{w}|}$ as $p(\mathbf{w}) = \prod_{i=1}^{|\mathbf{w}|+1} p(w_i \mid \mathbf{w}_{<i})$, where $w_{|\mathbf{w}|+1} = \text{EOS}$.

With this notation we will build first an n-gram LM for use in Chapter 7, and then the two types of neural baselines we use throughout.

4.2.1 Baseline n-gram LM

A "flat" hybrid word/character open-vocabulary n-gram model (Bisani and Ney, 2005) is defined over all strings Σ^+ from a vocabulary Σ with mutually disjoint subsets: $\Sigma = W \cup C \cup S$, where single characters $c \in C$ are distinguished in the model from single-character full words $w \in W$, e.g., $\underline{\text{a}}$ versus the word a. Special symbols $S = \{\underline{\text{EOW}}, \text{EOS}\}$ are end-of-word and end-of-string, respectively.

N-gram histories, i.e., the immediately preceding length-$n-1$ substrings conditioned on for factors $p(w_i \mid \mathbf{w}_{<i})$ are either word-boundary or word-internal (given whitespace left by the pretokenization used), i.e., $H = H_b \cup H_i$. String-internal word boundaries are always separated by the single whitespace character between tokens created through

[3] Even if we were to assume the set of characters closed, graphemes from other languages or other rare Unicode characters we have never seen during training may on rare occasion appear in random text samples. These are rare enough to not materially affect the metrics, but we need this UNK-like symbol to be prepared for them.

pre-tokenization.

As an example, if `foo`, `baz` $\in W$ but `bar` $\notin W$, then the string `foo bar baz` would be generated as: `foo` b a r EOW `baz` EOS. Possible 3-gram histories in this string would be, e.g., [`foo` b] $\in H_i$, [r EOW] $\in H_b$, and [EOW `baz`] $\in H_b$.

Histories $h \in H_b$ can generate symbols $s \in W \cup C \cup \{\text{EOS}\}$. If $s = \text{EOS}$, the string is ended. If $s \in W$, it has an implicit EOW and the model transitions to history $h' \in H_b$ for the next step. If $s \in C$, it transitions to $h' \in H_i$.

Histories $h \in H_i$ can generate symbols $s \in C \cup \{\text{EOW}\}$ and transition to $h' \in H_b$ if $s = \text{EOW}$, otherwise to $h' \in H_i$.

We use standard Kneser and Ney (1995) model training, with distributions at word-internal histories $h \in H_i$ constrained so as to only provide probability mass for symbols $s \in C \cup \{\text{EOW}\}$. We train 7-gram models, but prune n-grams with histories $h \in W^k$, for $k > 4$, i.e., 6- and 7-gram histories must include at least one $s \notin W$.

To establish the vocabularies W and C, we replace exactly one instance of each word type in the corpus with its spelled out version. Singleton words, i.e., words that only appear exactly once, are thus excluded from W, and character sequence observations from all types are included in training (a point we will pay much attention to in Part II of this thesis). Note that any word $w \in W$ can also be generated as a character sequence, so for perplexity calculation, we sum the probabilities for each way of generating the word, which is easy enough for this n-gram model, but not for the neural models we will describe next (necessitating the discussion in Section 4.2.3.3).

4.2.2 Neural model architectures

While neural language models can also take a hybrid approach (Hwang and Sung, 2017; Kawakami et al., 2017, but also Chapter 6 of this thesis), for this basic introduction we will restrict ourselves to simple single-level neural language models.

As sketched in Section 2.2, these models use neural networks to express how exactly the distribution $p(w_i \mid \mathbf{w}_{<i})$ depends on $\mathbf{w}_{<i}$. No matter whether they are Recurrent Neural Networks (RNN; Mikolov et al., 2010), Transformers (Vaswani et al., 2017), or even old neural n-gram models (Bengio et al., 2001), all can be abstracted into the same interface, again assuming d is some arbitrarily chosen embedding dimensionality:

- a space of cell states, $C \subseteq \mathbb{R}^{d'}$, where usually $d' > d$,
- an initial cell state, $c_1 \in C$,
- a projection, hidden$\colon C \to \mathbb{R}^d$, and
- a recurrence, advance$\colon C \times \mathbb{R}^d \to C$ that given a cell state and an embedding produces a new cell state,

where the functions and c_1 are learnable parameters.

Generation using this framework requires that each item in our vocabulary (be it words, subwords, or characters) is associated with a d-dimensional word embedding: starting with the initial cell state c_1, we generate a sentence autoregressively, at each timestep t extracting the "output" hidden state $h_t = \text{hidden}(c_t)$, sampling the lexeme w_t to be selected from the softmax over the dot product between hidden state and embedding $p(w_t \mid w_1, \ldots, w_{t-1}) = p(w_t \mid h_t) \propto \exp\langle h, e(w_t)\rangle$, and finally advancing the recurrence, setting $c_{t+1} = \text{advance}(c_t, e(w_t))$; repeating until the sampled lexeme $w_t = \textsc{eos}$.

The chosen language of cell and hidden state stems from the common RNN architecture Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997), but the same interface can be used for, say, Transformers (Vaswani et al., 2017) as well, as we will now sketch.

4.2.2.1 LSTM instantiation

A single-layer LSTM has cell states split into a memory cell[4] $m_t \in \mathbb{R}^d$ and a hidden state $h_t \in (0,1)^d$, making $d' = 2d$ in this single layer-case. The hidden state being so constrained will become clear once we consider the equations involved in both parts' calculation, i.e., the implementation of our functions hidden and advance in terms of four auxiliary variables $f_t, i_t, o_t, \tilde{c}_t \in (0,1)^d$ and learned parameters $W_f, W_i, W_o, W_m, U_f, U_i, U_o, U_m \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o, \mathbf{b}_m \in \mathbb{R}^d$, with all their indices denoting the forget, input, output, and memory gates, respectively:

$$
\begin{aligned}
[m_t, h_t] &= c_t \\
\text{hidden}(c_t) &= h_t \\
[m_{t-1}, h_{t-1}] &= c_{t-1} \\
f_t &= \text{sigmoid}(W_f x_t + U_f h_{t-1} + \mathbf{b}_f) \\
i_t &= \text{sigmoid}(W_i x_t + U_i h_{t-1} + \mathbf{b}_i) \\
o_t &= \text{sigmoid}(W_o x_t + U_o h_{t-1} + \mathbf{b}_o) \\
\tilde{m}_t &= \tanh(W_m x_t + U_m h_{t-1} + \mathbf{b}_m) \\
m_t &= f_t \odot m_{t-1} + i_t \odot \tilde{m}_t \\
h_t &= o_t \odot \tanh(m_t) \\
\text{advance}(c_{t-1}, x_t) &= [m_t, h_t]
\end{aligned}
$$

The input x_t is a word embedding $e(w_t) \in \mathbb{R}^d$ for the first layer of a model, but if we *stack* multiple layers of this recurrence higher layers take the hidden state h_t as their input to perform their own recurrence with their own state. The full cell state c_t as defined in our interface thus becomes a concatenation of individual layers' $[m_t, h_t]$, i.e., say, for a 3-layer model we have $c_t = [m_t^1, h_t^1, m_t^2, h_t^2, m_t^3, h_t^3]$ with the superscript denoting the layer

[4] It is more commonly called "cell" denoted c, but to avoid confusion with our interface we opt for this alternative name.

number. Our hidden state retrieval function would then be defined as $\text{hidden}(c_t) = h_t^3$, extracting only the final layer's output.

Note that this multi-layer case also allows for increased memory/hidden state cell dimensionality, provided the first and last layer are compatible with the d-dimensional word embeddings on either side, respectively, leading to non-square matrices W_* and U_*.

4.2.2.2 Transformer instantiation

The Transformer case is a bit more interesting as the recurrence is not quite as obvious or "baked-in" and a full definition of the architecture is not needed for our purposes.

The basic skeleton of a basic generative Transformer (i.e., Vaswani et al., 2017) is more akin to that of an n-gram model in that the history[5] is read in from scratch for each new prediction step. Its word embeddings are concatenated with positional embeddings that encode the position of each token in the sequence, and then fed into a series of Transformer encoder layers, each of which at its core uses a process called self-attention, which at its core in turn computes a weighted average of all tokens' current embeddings at the given layer, necessitating the preservation of positional information.

We can cast this into our interface by using the cell state to remember previous inputs' embeddings, the position we are currently at (to compute or retrieve appropriate positional embeddings), and output of the final layer at the last position. Note that the integral position as well as the potentially unbounded number of past inputs violate our claim that the cell state is a real-valued vector, but by positing a maximum context length (as is commonly done), we can still fit that overzealous definition well enough. That last output is what our function hidden will return given a cell state. Finally, our function advance uses all these ingredients to compute said final layer output and append the current embedding to the stored history as well as advancing the stored position.

[5]For Transformers the history need not be limited to any n per se given certain architecture choices but often is still capped but possibly at hundreds or even thousands of times as many tokens as for n-gram models.

4.2.3 Neural model granularities

Finally, after checking off pre-tokenization, we will, as promised, address what *tokenization* we will use for our neural language model baselines: characters for one kind of baseline, and subword units for the other.

4.2.3.1 Characters

As discussed in Section 3.6, recent advances indicate that full character-level modeling can be competitive with word- or subword-level modeling in certain settings. We use the LSTM-based implementation of Merity et al. (2018) with the char-PTB parameters for our character-level experiments unless stated otherwise.

4.2.3.2 BPE subword units

BPE-based open-vocabulary language models make use of sub-word units instead of either words or characters and are a strong baseline on multiple languages, as we will see in Section 6.4. Again, the set of subword units is finite and determined from training data only, but it is a superset of the alphabet, making it possible to explain any novel word in held-out data via some segmentation.[6]

One important thing to note is that the size of this set can be tuned by specifying the number of BPE **merges**, allowing us to smoothly vary between a largely word-level model (∞ merges) and a kind of character-level model (0 merges).

We use the BPE implementation of Sennrich et al. (2016) for all our experiments.

4.2.3.3 Warning: A BPE-based LM can output a string in any segmentation

Take the example sentence "The exoskeleton is greater". The commonly used reference BPE implementation of Sennrich et al. (2016) is entirely deterministic in the way it segments

[6] In practice, in both training and testing, we only evaluate the probability of the canonical segmentation of the held-out string, rather than the total probability of all segmentations, an issue discussed shortly in Section 4.2.3.3.

words into subword units, so let's us assume this "canonical" segmentation is "The ex|os|kel|eton is great|er".

This segmentation is the result of merges between smaller units. A model over subword units could however have produced the exact same sentence, simply by predicting such smaller units, e.g.: "T|he ex|os|kel|eton is gr|eat|er" or even "T|h|e e|x|o|s|k|e|l|e|t|o|n i|s g|r|e|a|t|e|r". We can see that what we are actually modeling in our PURE-BPE baseline is not the probability of a string, but the probability of a string and its chosen segmentation (i.e., the one that is actually created by the chosen BPE tokenizer). This ambiguity has been used as a regularization before by Kudo (2018), but in our application we do need the total probability of the string, so we would have to marginalize over *all possible segmentations*!

This is obviously absolutely intractable and so we are left with no choice but to simply approximate the probability of all segmentations as the probability of the tokenizer-output segmentation.[7]

[7]Alternatively, we could try restricting the model to only produce "canonical" segmentations, but this would require significant engineering logic that would be hard to parallelize and transfer onto GPUs for the caluclation of the vocabulary softmax.

Part II

Building open-vocabulary language models

Chapter 5

The INLM: Language modeling with an infinite vocabulary

In previous sections we hinted at the autoregressive modeling mechanism of usual language models as well as a neuralized recurrence mechanism that connects word types to word embeddings. In this chapter we will take a closer look at these workings and how they could in theory be used to define a model that rises to the challenge of open-vocabulary language modeling and the need to be prepared for *any* word by actually modeling an *infinite* vocabulary.

5.1 What if we could model latent lexemes?

We will propose a model that learns the appropriate units of meaning from an *infinite* set, elements of which we will henceforth call *lexemes*, by combining an autoregressive neural language model that predicts one lexeme at a time and a noisy channel in which these sequences of lexemes are transformed into the observed sentence, as well as a second neural language model that relates the spelling of each lexeme to its embedding.

By modifying the definition of the noisy channel we end up with different kinds of models: a model in which the lexemes correspond to infinitely many word types (as per our motivation) and the noisy channel just concatenates these lexemes with spaces, a model in which they correspond to subword units which are concatenated in the noisy channel, and

even a model in which lexemes correspond to arbitrary underlying forms that through the noisy channel give rise to noisy user-written text or inflected forms according to relevant morphological processes. For the purposes of this thesis, we will assume the first kind for simplicity, though the remaining challenges for meaningful noisy channels might not be as imposing a challenge if we can already solve that first case.

A model that recovers a sequence of underlying latent lexemes should boast not only good performance, but more importantly interpretability and usefulness for downstream tasks—flexibly biased towards or restricted to certain kinds of lexemes and processes, a variety of use cases can be met. The remainder of this document will detail how exactly to define such a model over an infinite set of possible lexemes to explain observed text, as well as give a sketch for one possible inference idea to be explored in the future.

5.1.1 ...didn't Bayesian Non-Parametric models already do this?

While the fully generative approach is shared by previous Bayesian models of language (e.g., Goldwater et al. (2006a)), even those that model characters and words at different levels (Mochihashi et al., 2009; Goldwater et al., 2011) have no embeddings and hence neither smoothing over similar words, nor any way to relate spelling to usage. They also have an impoverished model of sequential structure (essentially, n-gram models with backoff, i.e., some smoothing over similar contexts). We instead employ *recurrent neural networks* to model both the sequence of words in a sentence and the sequence of characters in a word type, where the latter sequence is conditioned on the word's embedding.

The resulting model is well-founded in linguistics and Bayesian modeling, but we can easily use it in the traditional non-Bayesian language modeling setting, assuming we can perform inference (which we will not show for the full model of Chapter 5, but for the simplified model of Chapter 6).

5.2 The INLM: neurally contextual distributions over infinite sets of discrete units

In this section, we propose the Infinite Neural Language Model (INLM, a name inspired by Beal et al. (2001)'s infinite HMM), a neural language model over an infinite set of types that incorporates a generative model of word spelling. That is, we aim to explain the training corpus as resulting from a process that first generated a lexicon of word types—the language's vocabulary—and then generated the corpus of tokens by using those types. Our Bayesian generative story combines a standard RNN language model (generating the word *tokens* in each sentence) with an RNN-based spelling model (generating the letters in each word *type*). These two RNNs respectively capture sentence structure and word structure, and are kept separate as in linguistics, connected only through words' embeddings which thus end up naturally inferred by combining evidence from type spelling and token context. We will elaborate on the merits of our design in Section 5.2.4 after formally establishing it.

5.2.1 Generating a lexicon

5.2.1.1 Connecting lexemes' spellings to their usage

The countably infinite set of the language's word types, which we henceforth call *lexemes* to avoid confusion, is the *vocabulary* $\mathcal{V}$—for simplicity we will refer to its elements w using natural numbers, i.e., $\mathcal{V} = \mathbb{N}$, though we will write them as ①, ②, ... This very explicitly does define an ordering, which will make sense shortly.

In our model, the observable behavior of a lexeme w is determined by two properties: a latent real-valued *embedding* $e(w) \in \mathbb{R}^d$ for some dimensionality d, which governs where w tends to appear, and w's spelling $\sigma(w) \in \Sigma^*$ (for some alphabet of characters Σ), which governs how it looks orthographically when it does appear.[1]

Our model captures the correlation between these, so it can extrapolate to predict the

[1]The exact nature of this governance depends on the aforementioned noisy channel, which we will ignore for now.

spellings of *unknown* words in any syntactic/semantic context. To illustrate this, consider this sample from the simplified model we will build in the next chapter, in which the words in `this font` were unobserved in training data, yet have contextually appropriate spellings:

> Following the death of Edward McCartney in `1060` , the new definition was transferred to the `WDIC` of `Fullett` .

To achieve this, we will posit a generative model of spellings that uses the embedding, $p_{\text{spell}}(\sigma(w) \mid e(w))$. A small neural language model itself, p_{spell} is given access to the embedding $e(w)$ and then proceeds to autoregressively predict the spelling $\sigma(w)$, terminating on an end-of-word token EOW.

Our intuition for this is that in most languages, the spelling of a word tends to weakly reflect categorical properties that are hopefully captured in the embedding. For example, proper names may have a certain form, content words may have to contain at least two syllables (McCarthy and Prince, 1999), and past-tense verbs may tend to end with a certain suffix. This is why $p_{\text{spell}}(\sigma(w) \mid e(w))$ is conditioned on the lexeme's embedding $e(w)$.

Armed with this new tool, we can describe the process of generating the lexicon.

Overall, the lexicon is then generated by a Pitman-Yor Process (PYP; Pitman and Yor, 1997)[2] that takes as a base distribution a mechanism that uses $\mathcal{N}(0, I)$ to sample an embedding for every atom and p_{spell} to sample a spelling for the atom given the freshly sampled embedding. As with probability 1 (wp1) no two draws from $\mathcal{N}(0, I)$ are the same, no two atoms (i.e., lexemes) drawn in the PYP are the same—though two lexemes can certainly share *spellings* (polysemy).

Let us formalize the generative story for the lexicon as follows. Given some hyperparameters $d \in [0, 1)$ and $\theta > -d$,[3] and starting with remaining probability mass $r := 1$, do

[2]Note that a simpler Dirichlet Process may be used, but the PYP has been shown to be more appropriate for natural language; more discussion on that can be found in Section 5.6.

[3]These are referred to as discount and strength, respectively; intuitively d makes later lexicon entries have

the following for all $w \in \{①, ②, \ldots\}$:

1. sample a proportion $p \sim \text{Beta}(1 - d, \theta + d \cdot w)$ (*stick-breaking* construction of the PYP), set the *prior probability* of the lexeme $p_w := r \cdot p$ and remaining mass $r := r \cdot (1 - p)$,
2. sample $e_{(w)} \sim \mathcal{N}(0, I)$, the d-dimensional embedding of the lexeme,
3. and sample $\sigma_{(w)} \sim p_{\text{spell}}(\sigma_{(w)} \mid e_{(w)})$.

The process and an example result of the process, a lexicon *stick* with infinitely many segments, are shown in Fig. 5-1.

Note that we technically also need an embedding $e(\textsc{eos})$ and reserve mass for a prior probability $p_{\textsc{eos}}$, but for simplicity of notation, we will ignore that harmless complication in the remainder of this exposition.

5.2.2 Generating running text using the lexicon

Using this lexicon, in particular p_w and $e_{(w)}$, we can now generate a sequence of lexemes using any neural language model we like. We will call this model the *transition dynamics* of the INLM.

The process is illustrated in Fig. 5-2. All that needs changing from our exposition in Section 4.2.2 for the infinite case is the prediction distribution (as summing over infinitely many exponentiated dot products as-is will diverge wp1). The solution is to multiply in the prior probability p_w for each lexeme:

$$p_{\text{dynamics}}(w \mid h) \propto p_w \cdot \exp\langle h, e_{(w)} \rangle$$

Note that we can interpret this in different ways:

- $\log p_w$ is the *bias* term that is often used in prediction models and in particular RNN language models, or

Beta distributions that skew more and more towards 0, yielding shorter segments and a slower decay, which as mentioned in Footnote 2 aids modeling natural language. Setting $d = 0$ recovers the Dirichlet Process in which all Beta distributions are the same.

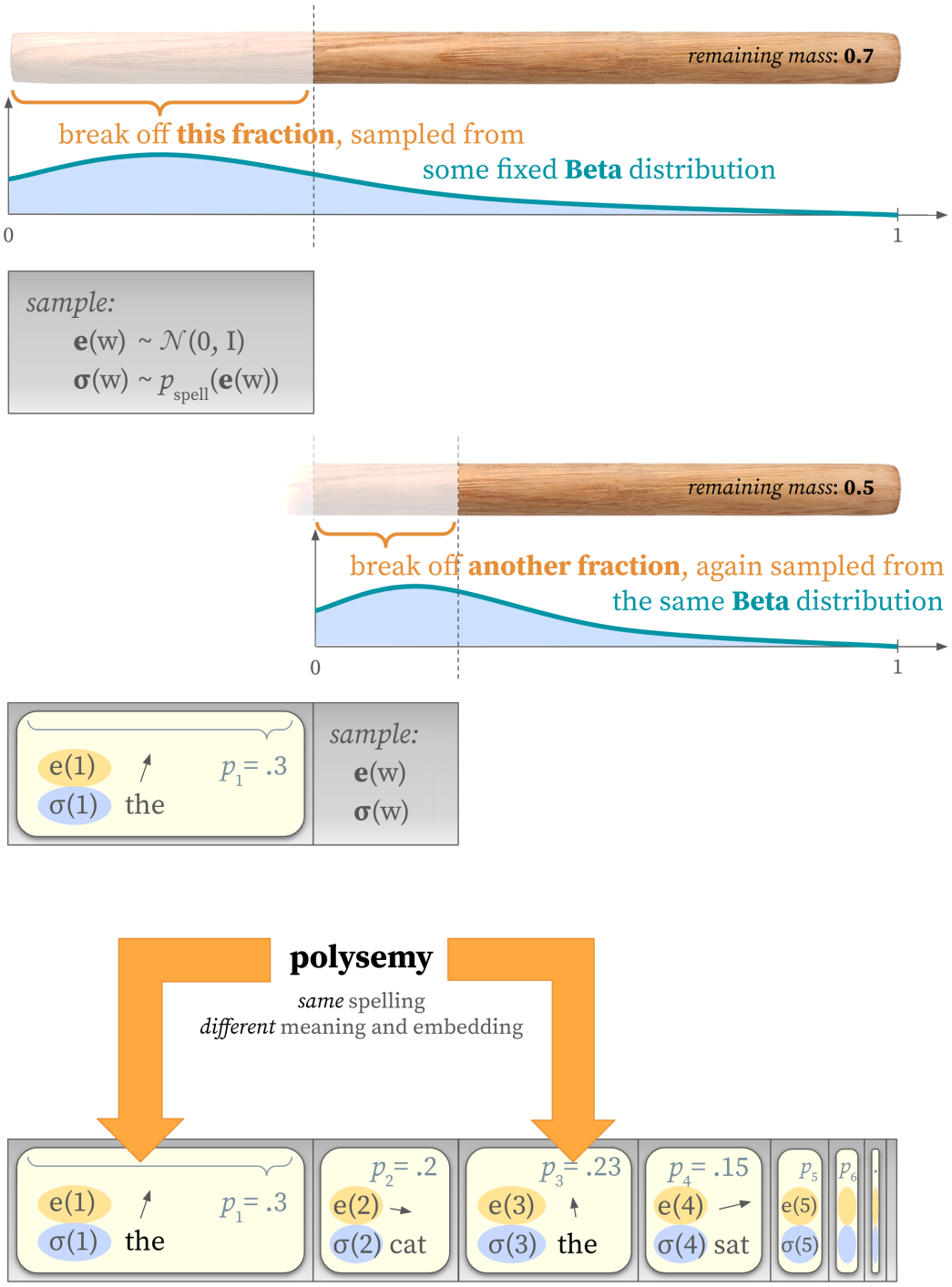


Figure 5-1. Top: sampling the first stick segment by breaking off a proportion sampled from a Beta distribution and labeling it using a drawn embedding and a speller-given spelling conditioned on that embedding. Middle: the process continues repeating this step, sampling a proportion of the *remaining* mass from the *same* Beta distribution. Bottom: the result is a stick of infinitely many segments, some of which may be labeled with the same spelling.

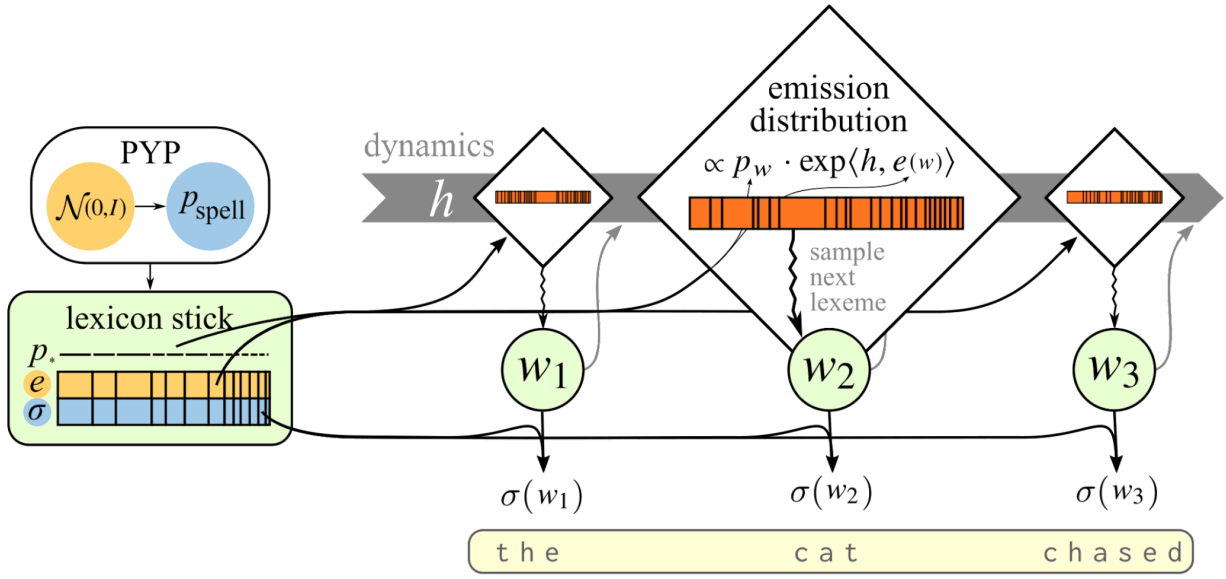


Figure 5-2. Generatng running text using the lexicon, distorting that "prior" stick into the emission stick that is the distribution generation samples from at each timestep.

- p_w is a prior over lexemes and $\exp\langle h, e(w)\rangle$ if interpreted as a likelihood $p(h \mid w)$ gives us a posterior of sorts over lexemes, namely p_{dynamics}, though this perspective is somewhat backwards to the usual intuition and doesn't fit with the overall result of $p_{\text{dynamics}}(w \mid h)$.

We will formally show in Section 5.2.3.1 that this distribution indeed has a finite normalizing constant wp1, but for now move on assuming as much.

The final missing step is to convert the sequence of lexemes $w_1, \ldots, w_n$ into an output sentence. As sketched earlier, to do so, we could either:

1. look up each lexeme's spelling $\sigma(w)$ and output the concatenation of these spellings (interspersed with spaces in the case where our lexemes are space-separated tokens, or using whatever notion of pre-token and detokenization the pre-tokenizer supplies) as our final output;

2. look up each lexeme's spellings $\sigma(w)$, but also transform each lexeme individually through some pre-specified noisy channel, i.e., a probabilistic string transducer

$t(\sigma' \mid \sigma(w))$, before outputing the concatenation of spellings σ';

3. proceed as in 1., but apply a specified noisy channel string transducer to the concatenated output as a whole.

In the latter two versions, where we do have a noisy channel that transforms the spelling "prototypes" into observed spellings, we might have, say, a finite-state transducer t that models a noise process or more complicated processes like morphological inflection—either individually (as in 2.) or by also modeling interactions between lexemes (as in 3.), in which case our lexemes might in fact be morphemes. In that sense, this work can be seen as an extension and expansion of earlier work on recovering morphological processes from running text with WFSAs (Dreyer and Eisner, 2011).

As it will turn out, our model and the inference method we sketch should be able to cover all three cases in theory, but, again, for simplicity, we will focus our efforts on the simplest of the three cases in this document and the formal specification of the full INLM, which we are now ready to do.

5.2.3 The full probabilistic model

The INLM specifies a joint distribution over the lexicon (e, σ) and an observed corpus of text[4], omitting the implied dependence of p_{spell} and p_{dynamics} on an additional set of finite

[4]This exposition again ignores the specialness of EOS to simplify notation.

parameters θ (that we may also want to place a prior on for regularization purposes):

$$p(e, \sigma, \text{corpus}) = \underbrace{\prod_{w \in \mathcal{V}} \left[\underbrace{\mathbb{P}_{\text{Beta}(1-d, \theta+d\cdot w)}\left(\frac{p_w}{\prod_{i=1}^{w-1}(1-p_i)} \right)}_{\text{stick-breaking PYP construction}} \cdot \underbrace{\mathbb{P}_{\mathcal{N}(0,I)}(e(w))}_{\substack{\text{prior on} \\ \text{embeddings}}} \cdot \underbrace{p_{\text{spell}}(\sigma(w) \mid e(w))}_{\substack{\text{spelling model} \\ \text{for all types}}} \right]}_{\text{lexicon generation}} \tag{5.1}$$

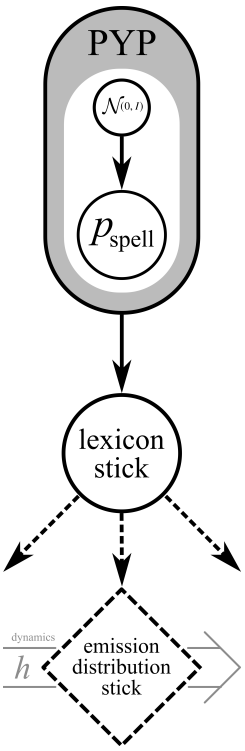
$$\cdot \underbrace{\sum_{n \in \mathbb{N}} \underbrace{\sum_{(w_1, \ldots, w_n) \in \mathbb{N}^n}}_{\text{lexeme sequences}} \left[\underbrace{\left(\prod_{i=1}^{n} p_{\text{dynamics}}(w_i \mid h_i) \right)}_{\text{lexeme-level dynamics}} \underbrace{p_{\text{channel}}(\text{corpus} \mid w_1, \ldots, w_n)}_{\text{noisy channel, e.g., concatenation}} \right]}_{\text{running text generation}}$$

where unfortunately the sum over all lexeme sequences[5] in the second line does not distribute over the timesteps because h_i depends on the lexemes and thus embeddings chosen in the path before.[6]

Visualizing the dependencies in a somewhat ad-hoc fashion, the important entities and quantities of the model are these:

[5]Note that if we choose the noisy channel to output lexemes' spelling as-is, interspersing spaces, n in the second line is fixed to the number of such space-separated units in text.

[6]If we had n-gram dynamics, we would end up joining paths again eventually, resulting in a lattice structure that admits dynamic programming, but in the general case, we could only *approximate* the entire trie of possibilities using a lattice (Buckman and Neubig, 2018).



The PYP prior generates our lexicon, represented as an infinite stick resulting from the stick-breaking construction of the PYP. Its parameters, a and b, we can search over or set sensibly.

The spelling model is used to label segments of that stick (i.e., lexemes). What does that mean? For every segment, we need to have the embedding be chosen to be predictive of the spelling; we model types, not tokens (more discussion on that in Section 5.2.4.1).

The (context-independent) lexicon stick (or "prior stick") is the distribution over lexemes, their spellings, and embeddings independent of position and hidden states of the RNN, as given by the PYP.

The emission stick at a timestep t is a *distorted* version of the lexicon stick that emphasizes lexemes whose embeddings fit the dynamics' hidden state h. It is what the generative model eventually samples a word from; from it we will read of the log-likelihood of heldout text to get our perplexity numbers.

5.2.3.1 Yes, we can (normalize)!

Let us settle the question we posed in Section 5.2.2: why would a distribution over this infinitely large set of lexemes, in particular the distorted emission stick, be normalizable?

Recount that the distribution has to normalize terms of the form $p_w \cdot \exp\langle h, e(w)\rangle$ for all $w \in \mathbb{N}$. For any given $h \in \mathbb{R}^d$, we will claim that the sum $\sum_{w\in\mathbb{N}} p_w \cdot \exp\langle h, e(w)\rangle$ is finite wp1 by showing that its *expectation* (w.r.t. random embeddings $e(w)$ distributed according to our prior) is finite.

Specifically, given that $e(w) \sim \mathcal{N}(0, I)$, we know that the dot product $\langle h, e(w)\rangle$, which essentially projects the isotropic Gaussian onto the vector h, follows a normal distribution $\mathcal{N}(0, \|h\|^2)$, so $\exp\langle h, e(w)\rangle \sim \text{LogNormal}(0, \|h\|^2)$ and since all embeddings are drawn IID, we have that $\mathbb{E}[\sum_{w\in\mathbb{N}} p_w \cdot \exp\langle h, e(w)\rangle] = \sum_{w\in\mathbb{N}} p_w \cdot \mathbb{E}[\exp\langle h, e(w)\rangle] = \sum_{w\in\mathbb{N}} p_w \cdot \exp\frac{\|h\|^2}{2} = \exp\frac{\|h\|^2}{2}$, which not only shows that indeed the expectation is finite as we wanted to show, but also gives us a sense of what magnitude to expect.

5.2.4 Why this structure and these priors?

So why construct a generative model of the lexicon that models the spellings of word types—like this? The basic intuition for combining spelling and context information is that in most languages, the spelling of a word tends to weakly reflect categorical properties that are hopefully captured in the embedding.[7] But beyond that most basic intuition, there are multiple ways to motivate the particular structure we chose.

5.2.4.1 A linguistic perspective

Duality of patterning[8] is a concept hypothesized to be a fundamental property of human language by Hockett (1960): the *form* of a word is logically separate from its *usage*. For example, while `children` may be an unusual spelling for a plural noun in English, it is listed as one in the lexicon, and that grants it all the same privileges as any other plural noun. The syntactic and semantic processes that combine words are blind to its unusual spelling.

In the INLM, this "separation of concerns" holds between p_{spell}, which governs word *form*, and p_{dynamics}, which governs word *usage*. We interpret this to mean that a word's distribution of contexts should be *conditionally independent* of its spelling, given its embedding: if we fully know the behavior of a word (a sequence of characters, morphemes, or phonemes) as encoded in its embedding, then its spelling is not going to give us any more information on its usage.

This does hold on the INLM, because p_{dynamics} does not consult spellings but only embeddings—but it would not hold true for a simple character-based language model, which blurs the two levels into one.

[7] For example, proper names may have a certain form, content words may have to contain at least two syllables (McCarthy and Prince, 1999), and past-tense verbs may tend to end with a certain suffix in English.

[8] This is related to Martinet (1949)'s concept of "double articulation," though it is not quite identical (Ladd, 2012).

The arbitrariness of the sign, relatedly, is Saussure (1916)'s idea that there is nothing about the *signified* (e.g., a cat) that would force it to be represented by any particular *signifier* (e.g., `cat`).

We agree with that notion, but only to some degree. Specifically, we believe that some pairings may be more likely *a priori* than others,[9] so while in our model, p_{spell} has support on all of Σ^*, allowing an embedding to in principle be paired in the lexicon with any spelling, it will still prefer some kinds of spellings to others (e.g., it is more likely that we use `cat` than `yxq`, and we may prefer `yeeting` to `yeeter` where an embedding that has high compatibility with contexts that call for gerunds was supplied).

Importantly, however, in contrast with most prior work that *compositionally creates* a word's embedding from its spelling (dos Santos and Zadrozny, 2014; Zhang et al., 2015; Kim et al., 2016), our model only *prefers* a word's embedding to correlate with its spelling, in order to raise the probability $p_{\text{spell}}(\sigma(w) \mid e(w))$. This preference is merely a regularizing effect that may be overcome by the need to keep the p_{dynamics} factors large, particularly for a frequent word that appears in many p_{dynamics} factors and is thus allowed its own idiosyncratic embedding–like the word `silly` that despite ending in `-ly` is not an adverb, the word `children` that does not look like a plural form, or the word `the` that would have us learn a very specific meaning for its letters and bigrams if we were to focus on constructing a so frequently-used embedding from its spelling using, say, a CNN.

In short, our spelling model is *not* supposed to be able to perfectly predict the spelling. However, it can statistically capture phonotactics, regular and semi-regular inflection, etc.

5.2.4.2 A modeling perspective

Very similar arguments can be made by starting from a language modeling perspective and thinking about how to build a statistically efficient model. In particular, the distinction

[9]Note that the spelling model is *not* able or supposed to be able to perfectly predict the spelling, but it is supposed to model a) morphological phenomena like inflection, b) certain regularities within semantic clusters (e.g., drug names), and finally c) general English phonotactics.

between word types (i.e., entries in a vocabulary) and tokens (i.e., individual occurrences of these word types in text) is also motivated by a generative (e.g., Bayesian) treatment of language: a lexeme's spelling is *reused* over all its tokens, not generated from scratch for each token.

This means that the term $p_{\text{spell}}(\sigma(w) \mid e(w))$ appears only once in (5.1) for each word type w—it is not raised to the 9999th power for a word that appeared 9999 times in the corpus. Thus, the training of the spelling model is not *overly* influenced by frequent (and highly atypical!)[10] words like `the` and `a`, but just as much as by rare words like `deforestation`.

As a result, p_{spell} learns how typical word *types*—not typical *tokens*—are spelled. It is worth pointing out that Baayen and Sproat (1996) even argue for using only the *hapax legomena* (words that only appear once in the training corpus) for predicting the behavior of rare and unknown words. The Bayesian approach (MacKay and Peto, 1995; Teh, 2006) which we follow here is a compromise: frequent word types are also used, but they have no more influence than infrequent ones.

To summarize, the usage of types instead of tokens is useful in predicting how *other types* will be spelled, which helps us regularize the embeddings of rare word types and predict the spellings of novel word types (a purpose for which we will use p_{spell} more explicitly in our finite simplification presented in the next chapter, detailed in Section 6.2).

5.3 Looking ahead: a finite simplification

To actually achieve real results, the next chapter will present a finite-vocabulary simplification of the INLM, which we call *Spell Once, Summon Anywhere* (SOSA). Looking ahead,

[10] The striking difference between types and tokens is very visible with `th`, which is the most common character bigram in words of the Penn Treebank as preprocessed by Mikolov et al. (2010) when counting word *tokens*, but only appears in 156th place when counting word *types*. Looking at trigrams (with spaces) produces an even starker picture: `␣th`, `the`, `he␣` are respectively the first, second, and third most common trigrams when looking at tokens, but only the 292nd, 550th, and 812th (out of 5261) when considering types.

we will express this model in a similar form as the INLM, again omitting the implicit dependency on neural network parameters and their regularization:

$$p(e, \sigma, s_1 \cdots s_n) = \tag{5.2}$$
$$\prod_{w \in \mathcal{V}} \left[p(e_{(w)}) \cdot p_{\text{spell}}(\sigma_{(w)} \mid e_{(w)}) \right] \cdot \prod_{i=1}^{n} p_{\text{dynamics}}(w_i \mid w_{<i}, e) \cdot \prod_{i:\, w_i = \text{UNK}} p_{\text{spell}}(s_i \mid h_i)$$

where $s_1, \ldots, s_n$ are the observed spellings that make up the corpus and $w_i = \sigma^{-1}(s_i)$ if defined, i.e., if there is a $w \in \mathcal{V}$ with $\sigma(w) = s_i$, and $w_i = \text{UNK}$ otherwise.

For the remainder of this chapter though, we will sketch an inference idea for the full INLM.

5.4 Inference using MCEM with semi-collapsed sticks

While it should in theory be possible to come up with an implementation of what we sketch in the sequel, even a fully realized instantiation and implementation of this sketch is likely infeasible to run on real natural language data, let alone large-scale data. That reasoning is why we have not seen this inference idea through to specific implementation and our claim that training the model "is possible" and "works this way" should be seen as optimistic theoretical consideration lacking empirical validation as well as definitional rigor. With that disclaimer, let us look into what it is we are proposing.

The conceptually simple way of training our model we choose to sketch is Monte Carlo Expectation Maximization (MCEM, Wei and Tanner, 1990): keep the *uncertainty over lexemes* with their embeddings (as there are *infinitely* many!), *maximize* over the *finite* set of the spelling and dynamics models' parameters. Specifically:

The E step concerns itself with building a distribution over the unobserved quantities given our model parameters. In our case this refers to our *lexicon stick*: embeddings and prior probabilities, i.e., lengths of stick segments, for all lexemes—though the RJMCMC-based sampler we will propose in this section will deal with "semi-collapsed" sample states

that represent *grounded delexicalized lexicon stick prefixes* (which we will explain momentarily in Section 5.4.1). In Monte Carlo EM, this means *sampling* such lexicon sticks given our RNN parameters.

The M step updates the RNN parameters given samples of the lexicon stick, i.e., all lexemes' embeddings and prior probabilities, using standard gradient descent methods to maximize the joint likelihood. Because this also gives us gradients with respect to the lexicon stick itself (which we can then use in the E step), we will start our exposition with this M step.

Before this though, let us define what exactly this distribution we will want to sample from is and what its samples (i.e., the lexicon representations which the M step will use) look like.

5.4.1 The sampler states: grounded delexicalized lexicon stick prefixes

Since every lexicon stick is an infinite object, we opt to instead represent a lexicon stick by its *grounded delexicalized lexicon stick prefix* (GDLP): a prefix of the full infinite stick where every segment w

- still has a prior probability p_w,
- may also have an embedding $e(w)$, and
- if it has an embedding, is *aligned to* (or *grounded in*) one or more observed tokens—which implies a spelling, but that spelling is not *explicitly* represented in the GDLP.[11]

As an example, consider observing text "the cat sat," i.e. $s_1 = \texttt{the}$. Then the stick

w	①	②	③	④	⑤	...
p_w	.1	.05	.08	.04	.005	...
$e(w)$						...
$\sigma(w)$	the	sat	cat	dog	the	...

[11]This restricts us to the case of the simplest noisy channel assumed so far.

is one of the infinitely many sticks consistent with the following GDLP:

w	①	②	③
p_w	.1	.05	.08
$e(w)$	↗	→	↖
tokens	{1}	{3}	{2}

This GDLP, however, is not only *more ambiguous* than the full lexicon stick in that it does not represent the infinite suffix of the stick that is not used to explain any observation, but it is also *more specific* because it specifies a single sequences of lexemes as an explanation for the observed sequence of spellings! For example, the stick we just gave is equally consistent with the following, different GDLP (highlighting the difference with colors):

w	①	②	③	④	⑤
p_w	.1	.05	.08	.04	.005
$e(w)$		→	↖		↗
tokens		{3}	{2}		{1}

Note that a GDLP has to align every token the sample of running text we perform inference on, but beyond that it may have and even end on[12] a number of "unused" segments like segment ① and ④ in our second example.

Note that a different approach with its own benefits and drawbacks would be to remove the explicit *ordering* of segments from our GDLP representation and try to marginalize over possible orders in likelihood calculations, but for now we opt for this explicit ordered representation.

GDLPs will be the samples we produce in the E step and use in the M step as follows.

5.4.2 The M step: backpropagation and gradient descent over RNN parameters

A given GDLP tells us exactly where and how the two RNNs p_{dynamics} and p_{spell} are used in producing the observed corpus. Every stick segment that has an embedding $e(w)$ and

[12]This is necessary to make the reordering jump we will propose in Section 5.4.3 reversible.

an (implied!) spelling $\sigma(w)$ means that the joint likelihood we seek to maximize contains a term $p_{\text{spell}}(\sigma(w) \mid e(w))$.

Similarly, the aligned tokens in the GDLP specify the embeddings that are used to advance p_{dynamics} and with the resulting hidden states, every token s_i multiplies in a factor $p_{\text{spell}}(w \mid h_i)$, assuming that s_i is aligned to lexeme w. That distribution is normalizable as we saw in Section 5.2.3.1, but unfortunately we still have no way of obtaining the normalizing constant Z in closed form. Even the fact that we know[13] Z_{finite}, the part of Z incurred by our finite prefix, and have a closed form expectation for Z_{infinite}, the remainder incurred by the (random) infinite stick suffix, does not help us much, as it is not $\mathbb{E}[Z]$ that is we are interested in to evaluate the likelihood of our running text, but $\mathbb{E}[-\log Z]$ (i.e., the log of a fraction with Z in the denominator)!

Approximating or bounding this term is one of the central challenges to be solved, but initial experimentation with a sample PYP (Fig. 5-3 illustrates draws from a PYP and Fig. 5-4 shows how finite sums become good enough quickly enough) suggested that the distribution of Z_{infinite} may in fact be approximated parametrically, either through an appropriate distribution family (Fig. 5-5 shows how the distribution over $\log Z$ looks lognormal after z-transforming, Fig. 5-6 shows the parameters of that lognormal distribution as a function of $\|\vec{h}\|$ are only somewhat linearly predictable, and using a linear prediction in Fig. 5-7 shows that it's not a great fit after all),[14] or, more simply and expressively by drawing a number of samples from the PYP a single time and using the resulting samples with a dummy hidden state (as rotation does not matter given our isotropic prior and we can use the magnitude of any specific hidden state to rescale the result with the dummy).

With all that in place, we should be able to maximize the resulting joint likelihood by performing backpropagation on the expected negative log likelihood, yielding not only gradients for the parameters of our RNN models (which we can use to perform

[13] Prefix segments without embeddings actually do not produce a deterministic contribution to Z_{finite}, but we have a closed distribution for this finite amount that we can use to compute our desired expectation.

[14] We could use fixpoint solving like for geometric series to find such an approximation from few samples.

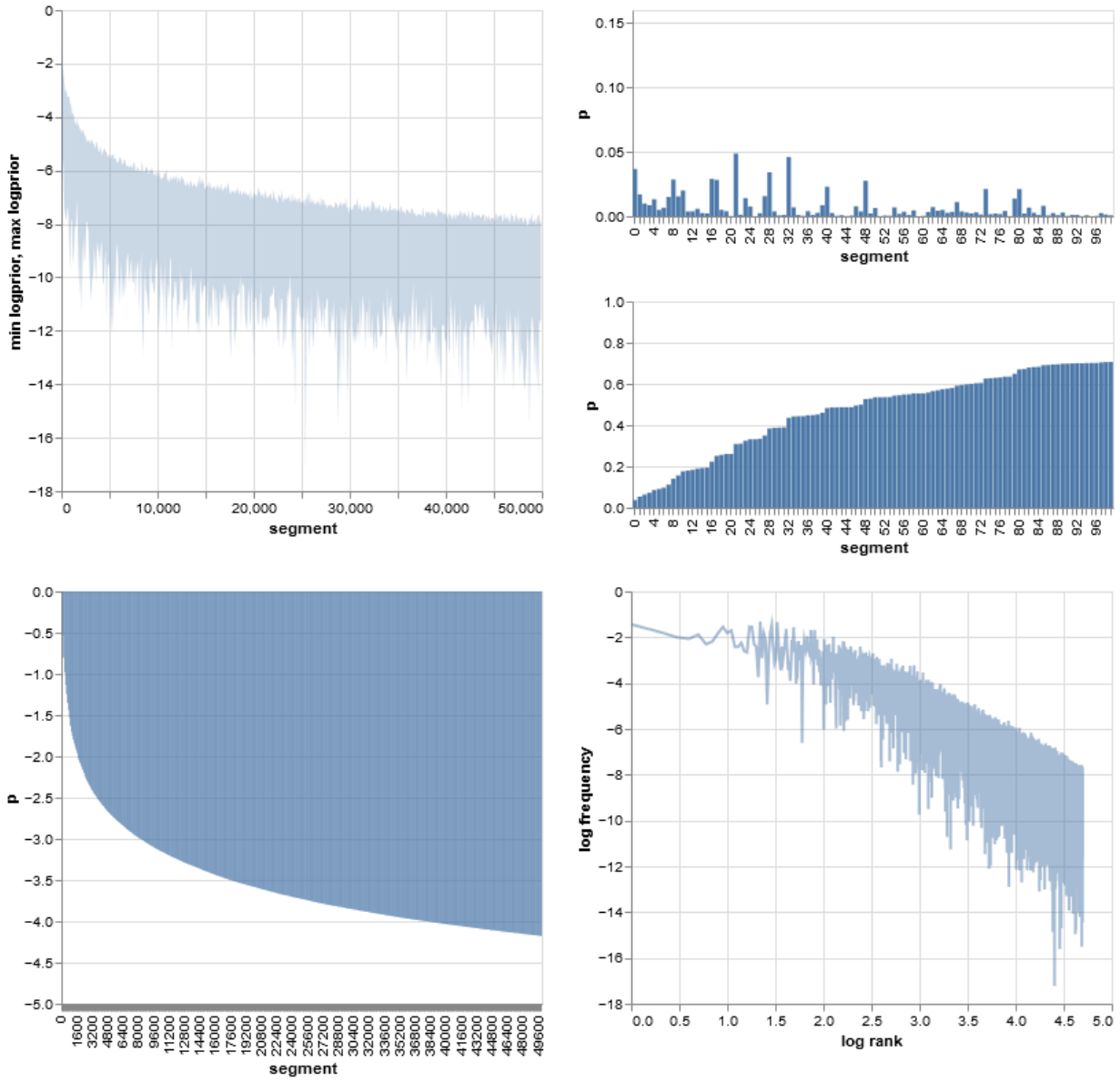


Figure 5-3. A draw from a PYP with $\theta = 40$, $d = 0.4$. After drawing 50k items, the stick still reserves ~ 0.00007 probability mass. Top left: all 50k draws, log probability of the individual segments bucketed for visualization, the shaded area being between the smallest and largest item in a given bucket. Top right: individual segment probabilities and cumulative probability for the first 100 segments drawn. Bottom left: log remaining probability mass on the stick through all 50k draws. Bottom right: log frequency/probability of the individual segments plotted against the log-rank.

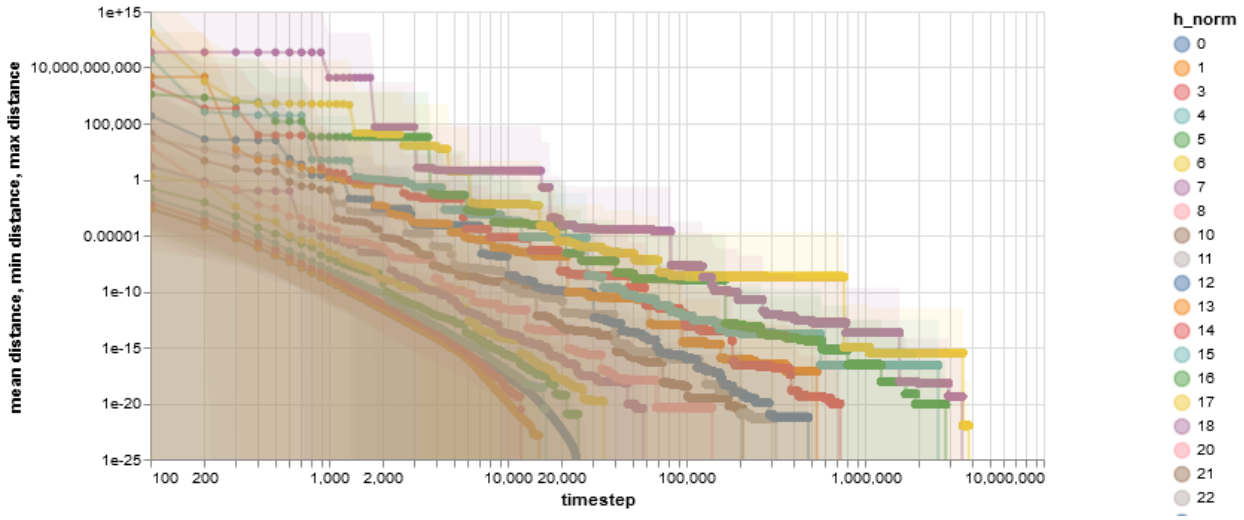


Figure 5-4. Varying $||\vec{h}||$ from 0 to 23 (directionality does not matter as the Gaussian embeddings are drawn from is isotropic) forms 24 different lines (mean with surrounding shaded areas indicating min/max over 10k different draws) that each show how as a function of segments already drawn, the difference between the finite sum and the "final" Z (as in the Z at the end of all 10,000,000 draws) gets smaller and smaller for a PYP with $d = 10$ and $\theta = 0.1$ for even faster convergence than the one shown in Fig. 5-3, reaching 99% allocated probability after just 75 draws. Larger values of $||\vec{h}||$ correspond to lines starting higher and reaching small values later, understandably given the larger variance involved.

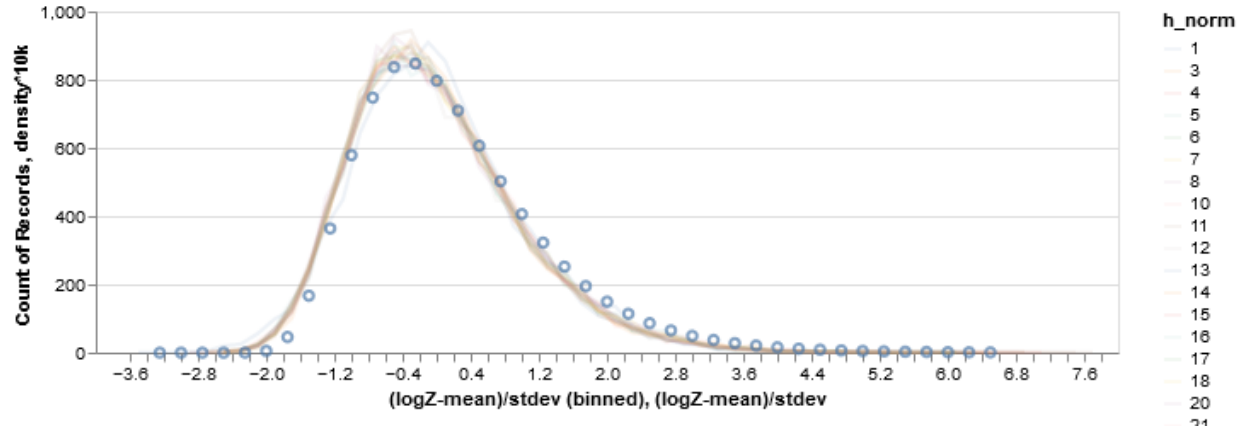


Figure 5-5. Looking at the distribution of the "final" $\log Z$'s from Fig. 5-4, we can see that after z-transforming, all values of $||\vec{h}||$ yield very similar distributions (faint lines) that closely fit a lognormal distribution (circles) with shape $1/e$, location $-e$, and scale e.

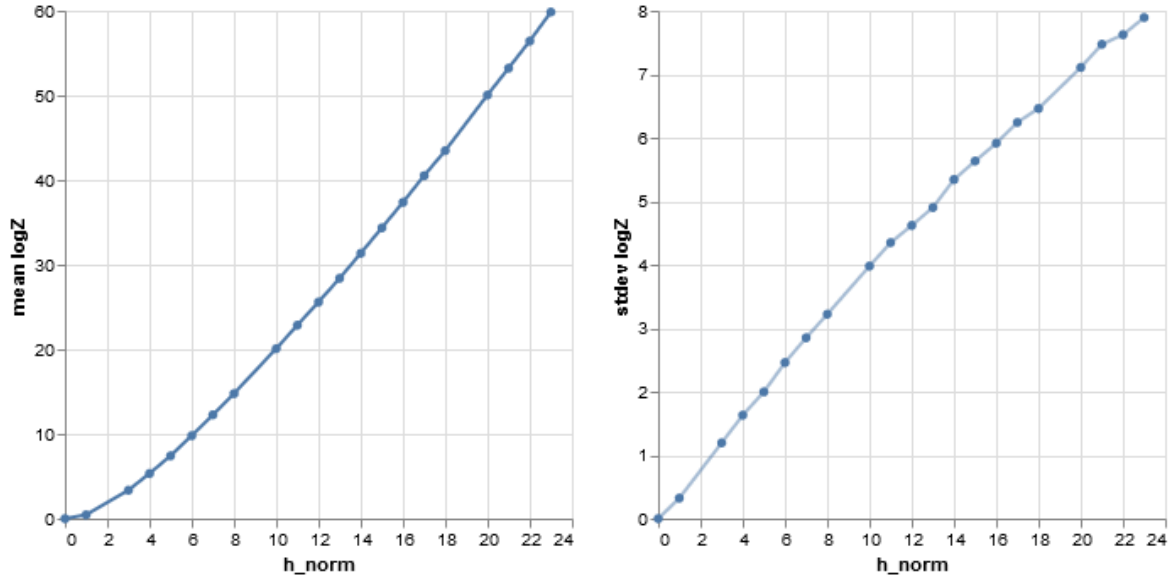


Figure 5-6. How are the parameters of the individual lognormal distributions in Fig. 5-5 for each value of $||\vec{h}||$ predictable from $||\vec{h}||$ itself?

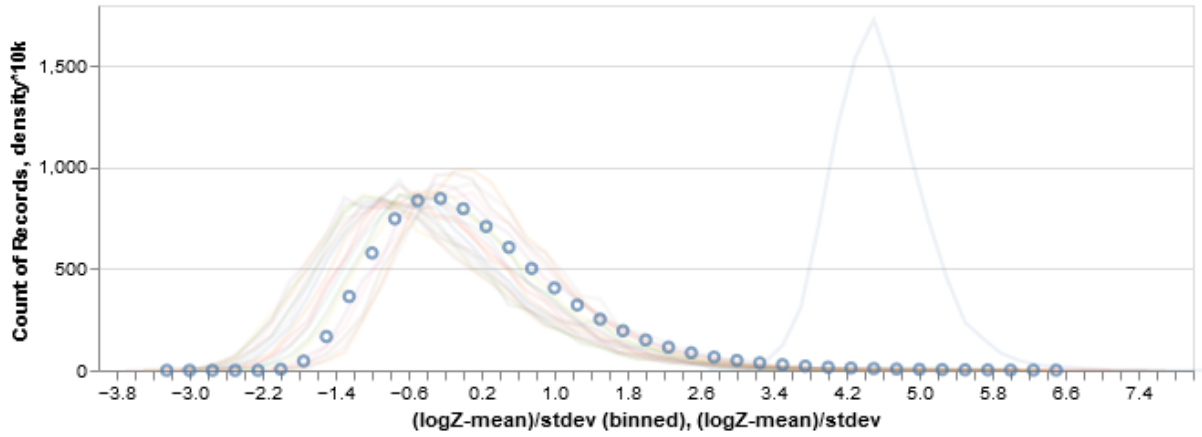


Figure 5-7. A similar plot to Fig. 5-5, except the faint lines corresponding to values of $||\vec{h}||$ now come from predicting the transformation from a parametric lognormal approximation using a linear predictor fit to the plots in Fig. 5-6, showing that that linear relation is not good enough.

gradient descent and conclude the M step), but also for the used embeddings $e(w)$ and prior probabilities p_w. These gradients will prove useful in the more complicated E step.

5.4.3 The E step: semi-collapsed RJMCMC with gradient information

We propose to sample new GDLPs through a Markov Chain Monte Carlo (MCMC; Metropolis et al., 1953) scheme, meaning that all we need to specify is a way to obtain *unnormalized* joint likelihoods for a GDLP and various jump proposals that can be accepted or rejected through a Metropolis-Hastings (MH; Hastings, 1970) step.

The former, the joint likelihood, consists of four terms:

- the prior probability of the lexemes' prior probabilities p_w under our PYP (easy, evaluate our finite prefix's density using the stick-breaking construction),
- the IID prior probabilities of embeddings of all segments (easy, we have the Gaussian density),
- the independent probabilities of generating corresponding spellings from the embeddings of all segments that actually have a token associated with them (easy, since our speller is a locally normalized language model this is the usual language modeling loss; same as in the M step), and finally
- for each token in running text, the probability that the aligned segment was actually sampled from the logprior-weighted softmax given the hidden state at that timestep. This last step corresponds to computing the standard language modeling loss, which, again, is unfortunately hard, requiring calculation or approximation of $-\log Z$ at each step, exactly as in the M step.

Once we settle on a way to calculate that last term as for the M step, the likelihood should not pose any conceptual trouble.

For jumps, there are a few simple proposals we can start with:

- change prior probabilities p_w (through changing the proportions involved in the stick-breaking construction, for example by resampling from a Dirichlet centered on the current proportions),
- resample embeddings $e(w)$ (we can start with very naive multivariate Gaussian jumps, but the M step gradients will be useful, see the note below this list).
- switch token alignments between segments that are associated with the same spelling (while making sure none of these segments are left unaligned to assure they all get to keep their embedding, we will deal with other reassignments later),
- reorder segments (sampling a new order by means of a size-biased permutation using the prior probabilities; if we are careful about including the infinite suffix in the calculations, this may even result in moves that are *always* accepted, a case of Gibbs sampling (Geman and Geman, 1984)), and finally of course
- any combination of the above.

For jumps that change prior probabilities or embeddings, the gradients obtained in the M step will prove very useful: we can use them in an Stochastic Gradient Langevin Dynamics (SGLD; Welling and Teh, 2011) scheme to obtain good proposals.

However, we also need to allow the model to allocate new segments in the GDLP, essentially providing new lexemes that tokens can align to—and in return we need jumps that remove unused segments. The trans-dimensionality inherent in such moves leads to our MCMC scheme turning into Reversible Jump MCMC (RJMCMC; Green, 1995), adding another set of jumps, organized as symmetric pairs:

- remove the last segment of the GDLP if it has no embedding associated with it / add a segment at the end with no embedding associated with it,
- re-align one or more tokens to a segment that has no embedding and no other tokens aligned with it (only existing segments, see prev. item), sampling a new embedding

(possibly from a smarter proposal than just a standard normal distribution) in the process / align all tokens of some segment to other segments of the same spellings, destroying the embedding of that segment in the process.

Armed with these jumps and whatever way to estimate the joint likelihood we settle on in the M step, all we need is an initial state, which could very simply assume no polysemy, i.e., one segment per spelling, create stick segment lengths and order from the spelling's frequency in the corpus, and embeddings either from some pretrained language or word embedding model or simply randomly initialized.

While all this in theory should yield a complete inference method if spelled out (e.g., by showing detailed balance for the RJMCMC steps only sketched here), in practice even this "semi-collapsed" sampler that already avoids pitfalls of a naive sampler may mix terribly slowly and scale poorly or rather not at all to real data.

Introducing smarter moves that for example merge or split segments with identical associated spellings may help ameliorate this issue to a degree. For this, we may find it useful to consider the ideas proposed by Salesky et al. (2018), explained in Section 3.5.2. While the task of gradually growing a BPE vocabulary may not be directly related, the approaches proposed to initialize the embeddings of the newly introduced types (random, averaging, or the latent code of an autoencoder that predicts both original types' embeddings) may be precisely what we want to do ourselves here.

In the long run, however, it will be worth thinking about other ways to perform inference that may be computationally simpler, perhaps at the cost of introducing possibly biased approximations. For example: can we try to use a collapsed scheme like the Chinese Restaurant Process that is generally used for PYPs, somehow correcting for the wrongly assumed exchangeability (which we lose because of our neurally distorted emission sticks)?

5.5 Evaluation proposals on synthetic and simplified data

Again, limiting ourselves to the case where the spellings we observe are not noisy, we posit a number of experiments that (if this model were actually implemented) could show us whether indeed this model structure allows us to learn something interesting.

5.5.1 Can we "recover" small-scale "handcrafted" synthetic INLMs?

To verify that our inference algorithms work, we can manually specify some arbitrary INLM, meaning that we choose some dynamics and speller models, then sample either one or many lexicon sticks using the process described in Section 5.2.1, and with those lexicon sticks, sample running text as described in Section 5.2.2. Given either just raw strings, or perhaps already found units (no noisy channel) or even types (i.e., clustering of tokens with the same spelling, defining how many lexemes will be needed), do our inference algorithms "recover" models and lexicon stick posteriors that are "similar" to the original ones? Of course, this notion of similarity must be formalized depending on what exactly it is we want to test. In either case, these recovery experiments are closer to unit tests than to actual uses of the model.

5.5.2 Experimenting on simplified real data

A text of full words with arbitrary spellings is likely too big for the barely collapsed sampler we proposed, so we propose three simplified scenarios for modeling real text.

POS tags Transforming text by replacing each word with its part-of-speech (POS) tag, we can build an INLM that models this text using lexemes whose spellings are now in the finite set $\{\texttt{NOUN}, \texttt{VERB}, \ldots\}$ of chosen POS tags, meaning that p_{spell} can be greatly simplified into a simple categorical distribution, still parameterized as a neural network that takes in an embedding. Besides being computationally simple, this setup is also appealing as

it allows us to analyze the resulting lexemes as clusters of original tokens within a given POS tag. Do fine-grained distinctions (e.g., a segment spelled `NOUN` that is aligned with common nouns in the original text vs. one aligned with proper nouns) emerge when we choose a more coarse-grained set of POS tags (which may only have one tag `NOUN` for both)?

Consonant-Vowel sequences As a next step, we may consider as possible spellings the set {`C`, `V`}*, mapping all consonants in a word to `C` and each vowel to `V`, i.e., modeling the string "the cat sat" as `CCV CVC CVC`.[15] This will require an autoregressive spelling model over this binary alphabet and (unlike with real data) it is not too unlikely that just sampling from this spelling model we would obtain spellings that could cover all those that occur in our data. For evaluating the model, we could again check whether lexemes represent interesting clusters of the original word types that we simplified.

Upgoer5 data The final step short of running the INLM on full data is to run it on a subset of natural language sentences constrained to a small vocabulary to constrain the space of spellings that the lexemes can take and make inference with even naive and inefficient algorithms realistic.

5.6 Related work

In this section we will not reurgitate the complete survey of related work found in Section 3.3 and Section 3.4, but highlight the connections between the ideas of this chapter and select references.

As laid out in Section 3.3.1, years ago, the influential papers of Kim et al. (2016) and Ling et al. (2015) first combined the two layers of characters and words by deterministically constructing word embeddings from characters. However in doing so, they train the

[15] A smarter transformation that realizes "th" is a single consonant could also be used.

embedding function on tokens, not types, to "get frequent words right"—ignoring the issues discussed in Section 5.2.4. Likewise, the "character-skipping" models in Section 3.4.1 (e.g. Chung et al., 2017) can be understood to essentially *construct* word embeddings from characters, again overemphasizing learning frequent spellings (Section 5.2.4.2). Essentially, these models all still have to re-generate a word every time it is encountered, or put differently, they cannot *memoize*.

Most obviously relevant to our proposal is the work discussed in Section 3.4.2, where the idea is to learn segmentations for text by marginalizing over possible segmentations, either by approximating (e.g. Buckman and Neubig, 2018) or through smart independence assumptions, i.e., model structure (e.g. Kong et al., 2016). In a sense this is precisely what we are trying to do but without the limitations of either approaches (approximation and independence assumptions). Nevertheless either route of making this process of marginalization feasible even with our more complex model may be a fruitful avenue for research.

The perhaps most interesting line of work and in a way the most crucial one to connect to, are the pre-neural language models that use Bayesian Nonparametrics to handle infinities in context length and vocabulary both, introduced in Section 3.4.3. Of particular relevance are researchers' attempts to try and incorporate character information to not only explain word use and re-use, but also word formation (Goldwater et al., 2006a, 2009; Mochihashi et al., 2009; Goldwater et al., 2011; Andrews et al., 2017; Johnson et al., 2007). A very exciting and promising approach, it led to interesting uses in unsupervised word segmentation (learning units as we would like to) and indeed is a significant spiritual predecessor for our work.

As hinted at early in this chapter in Section 5.1.1, the central weakness with all these previous models however is the impoverished model of sequential structure: n-gram backoff models (Goldwater et al., 2006a; Andrews et al., 2017) or context-free models (Johnson et al., 2007) to model both the sequence of words in a sentence and the sequence

of characters in a word type. The limitation here, importantly, is not necessarily that of a finite n, because, again, the unbounded hierarchy of, say, the sequence memoizer in fact allows for infinite context, but instead the issue is the *fixed* and somewhat linear[16] backoff order: a 3-gram distribution depends on the 1-gram distribution *through* the 2-gram distribution, it can not make use of *arbitrary features* of the history. For all its similarity to two-level models like Goldwater et al. (2009),[17] our model on the other hand uses an RNN to incorporate dependence on history into the PYP-based lexicon model. Thus taming infinite context dependence through finite parameters, neural models allow for arbitrarily complex features and interactions in generation history through the use of embeddings (Turian et al., 2010), through which we can even nicely relate spelling and usage.

In conclusion, we could say that the INLM is trying to bridge neural modeling and nonparametrics: add an infinite vocabulary and the story of its generation to autoregressive neural models, freeing character-level components from being restricted to tokens[18]—or add the free and complex interactions with the history word embeddings and carried hidden states allow to autoregressive nonparametric models.

In the next chapter, we will put this idea to the test by simplifying this hypothetical "holy grail" model to be finite-vocabulary allowing us to train and evaluate on larger-scale natural text.

[16] It is worth noting that Wood and Teh (2008) propose a multi-floor Chinese Restaurant Process to explicitly cover different backoff as a mixture of the aforementioned models, but again these orders have to be manually specified and are thus limited.

[17] De Marcken (1996) presented a similar approach using the terminology of Minimum Description Length.

[18] Indeed, we will find a way to achieve the second part of this goal still within a finite framework in the next chapter.

Chapter 6

Spell Once, Summon Anywhere: A *Practical* Two-Level Open-Vocabulary Language Model

Published research and text

This chapter is largely taken from Mielke and Eisner (2019).

To test whether this idea of connecting spellings and usage through embeddings is not only sensible in theory but also useful in practice, we will in this section consider a greatly simplified model that is open-vocabulary while operating at its core over a fixed and finite set of word types.

On a high level, we still separate the generation of the lexicon from the generation of running text using the lexicon, but in this chapter we will assume we know *the set of lexemes* and in fact *observe these lexemes and not just possibly-ambiguous spellings* to keep things simple, i.e., tokens in data are labeled as to whether they are the same lexeme based on whether they are spelled the same.[1] With this elimination of uncertainty over the lexicon entries and their order as well as their assignment to tokens in running text, it becomes easy to perform MAP inference in a thus simplified model.

To turn this finite model into an open-vocabulary model we will include an UNK type,

[1]We could make the point that this weakness is largely negated by today's commonplace Transformers that are all about contextual embeddings and thus should be able to perform, say, sense disambiguation just fine.

but whenever it is predicted, *actually spell out the word* using our generative spelling model—because just as p_{spell} has an opinion about how to spell rare words, it also has an opinion about how to spell novel words.[2]

Using this model, we beat previous work and established baselines, reaching (at the time of the paper's publication) state-of-the-art results on multiple datasets. We will also ablate this complete model and zoom in on various quantitative as well as interesting qualitative results.

We begin by concisely stating a first, closed-vocabulary, version of our model in Section 6.1. Section 6.2 then motivates and describes a simple way to extend the model to the open-vocabulary setting, with Section 6.3 providing details on how to train this model. Section 6.4 contains quantitative and qualitative experiments on multiple language modeling datasets, with implementation details provided in supplementary material. Finally, we clarify the relation of this model to previous work in Section 6.5 and summarize our contribution in Section 6.6.

6.1 A joint model of a finite lexicon and fully observed text

6.1.1 Preliminary restrictions to simplify modeling

We now assume that a language's word types, which we will keep calling *lexemes*, are discrete elements w of the *vocabulary* $\mathcal{V} = \{①, ②, \ldots ⓥ\}$ that now is very much finite.

Furthermore, we will model text that has already been tokenized, i.e., it is presented as a sequence of word tokens.

In fact, we assume that there is a 1-to-1 known correspondence between the observed word tokens and the language's lexemes (namely $\sigma(\cdot)$). In other words, this chapter makes

[2] Luong and Manning (2016) before the re-discovery of BPE proposed an open-vocabulary neural machine translation model in which the prediction of an UNK triggers a character-level model as a kind of "backoff"—just like here! This section will not only provide a proper Bayesian explanation for this trick but also find that it is insufficient (training the speller model only on unknown words performs significantly worse than training on both known and UNKed words): training on types, as suggested in Section 5.2.4.2, is more effective for the task of language modeling.

the very common simplifying assumption that the training corpus has *no polysemy*, so that two word tokens with the same spelling always correspond to the same lexeme. We thus assign distinct lexeme numbers ①, ②, . . . , ⓥ to the different spelling types we find in our training corpus (the specific assignment or order does not matter in the simplified version of this chapter and is merely an implementation detail).

This allows us to claim that we not only observed the spellings $\sigma(①), \sigma(②), \ldots, \sigma(ⓥ)$ of these v assumed lexemes in our lexicon, but also observed the actual token or lexeme sequence $w_1, \ldots, w_n$ where each w_i is a lexeme number.

6.1.2 The closed-vocabulary model

We will again use e and σ to refer to the functions that map each lexeme w to its embedding and spelling, making the full lexicon specified by (e, σ).

To simplify the exposition, in this first subsection our model will not be open-vocabulary yet (we will extend it towards that in Section 6.2, formalized in Eq. (6.2)). Given fixed vocabulary size v and text length n, the model specifies a joint distribution over the lexicon and corpus:

$$p(\theta, e, \sigma, w_1, \ldots, w_n) \;=\; p(\theta) \cdot \underbrace{\prod_{w \in \mathcal{V}} \Big[\underbrace{p(e(w))}_{\substack{\text{prior on} \\ \text{embeddings}}} \cdot \underbrace{p_{\text{spell}}(\sigma(w) \mid e(w))}_{\substack{\text{spelling model} \\ \text{for all types}}} \Big]}_{\text{lexicon generation}} \cdot \underbrace{\prod_{i=1}^{n} p_{\text{dynamics}}(w_i \mid \vec{w}_{<i}, e)}_{\substack{\text{lexeme-level} \\ \text{recurrent language model} \\ \text{for all tokens}}} \tag{6.1}$$

where p_{LM} (Section 6.3.4) and p_{spell} (Section 6.3.5) are the RNN sequence models from Section 5.2 that are parameterized by θ[3] (the dependence is again omitted above for space reasons), and $w_1, \ldots, w_n$ is the observed sequence of lexemes.

As we can see, the generative story for the observed training corpus thus again follows the same two steps, this time more simple than in the full INLM:

[3]Essentially, θ describes the patterns of sequential structure in the language's words and sentences, while e and σ describe possibly idiosyncratic lexical resources that can be summoned by p_{dynamics} using θ.

Generate the structure of the language. First draw parameters θ for both RNNs (from a spherical Gaussian). Then draw an embedding $e(w)$ for each lexeme w (from another spherical Gaussian).[4] Finally, sample a spelling $\sigma(w)$ for each lexeme w from p_{spell}, conditioned on $e(w)$ (and θ).

Generate the corpus. In the final term of (6.1), generate a sequence of lexemes $w_1, \ldots, w_n$ from p_{dynamics} (using its parameters in θ). Applying σ deterministically yields the actually observed training corpus, spelled out as a sequence of spelled words $\sigma(w_1), \ldots, \sigma(w_n)$.

So, working through this generative story in reverse, given the observed sequence, we train θ and e jointly by MAP estimation: in other words, we choose them to (locally) maximize (6.1).[5] This is straightforward using backpropagation and gradient descent (implementation and hyperparameter details in Section 6.3).

6.2 Open vocabulary by "spelling" UNK

Our parametric and finite model only includes parameters for finitely many different words—the vocabulary. We seek a model that can nonetheless assign a well-defined positive probability to any sentence, even if it includes out-of-vocabulary ("unknown") words—without escalating to taming infinity explicitly in the model as with the INLM of the previous chapter. The problem is that here out-of-vocabulary words do not list any embeddings or spellings in our lexicon. So how do we know when to generate them and how to spell them?

[4] We will later find the MAP estimates of θ and $e(w)$, so the Gaussian priors correspond to L_2 regularization of these estimates, a.k.a. weight decay.

[5] The non-Bayesian view on this is that we maximize the *regularized likelihood* of the model given the observations.

6.2.1 The solution: use the spelling model!

Recall the most popular traditional way of dealing with unknown words: the introduction of a new UNK type that represents all rare and unknown words.[6] The story usually ends there, but in contrast to a lot of this previous work we have an explicit spelling model to use that comes in very handy to go beyond just giving up at every UNK. The reason for that is that just as p_{spell} has an opinion about how to spell rare words, it also has an opinion about how to spell novel words.

This allows the following trick. We introduce the special lexeme UNK so that the vocabulary is now $\mathcal{V} = \{\text{UNK}, ①, ②, \ldots, ⓥ\}$ with finite size $v + 1$ and refine our story of how the corpus is generated. First, the model again predicts a complete sequence of lexemes $w_1, \ldots, w_n$. In most cases, w_i is spelled out deterministically as $\sigma(w_i)$. However, if $w_i = \text{UNK}$, then we spell it out by sampling from $p_{\text{spell}}(\cdot \mid \vec{e}_i)$, where $\vec{e}_i$ needs to be an appropriate chosen embedding for this novel word, explained below. The downside of this approach is that each UNK token samples a fresh spelling, so multiple tokens of an out-of-vocabulary word type are treated as if they were separate lexemes.

6.2.2 Which embedding to feed?

Recall that the spelling model generates a spelling *given an embedding*. So what embedding $\vec{e}_i$ should we use to generate this unknown word? Imagine the word had actually been in the vocabulary. Then, if the model had wanted to predict that word, $\vec{e}_i$ would have had to have a high dot product with the hidden state of the lexeme-level RNN at this time step, $\vec{h}$. So, clearly, the embedding that maximizes the dot product with the hidden state is just that hidden state itself.[7] It follows that we should sample the generated spelling $s \sim p(\cdot \mid \vec{h})$, using the current hidden state of the lexeme-level RNN p_{dynamics}.

[6] It is also possible to take this idea further and introduce or multiple UNK types representing classes of rare and unknown words defined by features of the character sequence (e.g., Klein and Manning, 2003, *inter alia)*, but we will not require such finesse.

[7] At least, this is the best $\vec{e}_i$ for which $||\vec{e}_i|| \leq ||\vec{h}||$ holds.

6.2.3 Warning: token generation is technically ambiguous

With this approach, we have to be aware of very subtle issue of spurious ambiguity: an in-vocabulary token can now be generated in *two* ways, either as the spelling of a known lexeme or from spelling out UNK. Formally, a spelling s that is actually associated with a lexeme in $\mathcal{V} \setminus \{\text{UNK}\}$, i.e., for which there is some $\textcircled{i} \in \mathcal{V} \setminus \{\text{UNK}\}$, could now be generated two ways: as $\sigma(\textcircled{i})$ and as $p(s \mid \vec{h})$. This complication technically turns our model into a latent-variable model (which would make inference much more complicated). To remedy this, we could explicitly set $p(s \mid \vec{h}) = 0$ for any s for which there is some $\textcircled{i} \in \mathcal{V} \setminus \{\text{UNK}\}$ with $\sigma(\textcircled{i}) = s$ and any $\vec{h}$, which would require us to renormalize, either globally by

$$p(s \mid \vec{h}) = \frac{p_{\text{spell}}(s \mid \vec{h})}{1 - \sum_{w \in \mathcal{V}} p_{\text{spell}}(\sigma(w) \mid \vec{h}))}$$

or locally by disallowing EOW if the string generated so far is in fact a word in the vocabulary (which is the arguably hackier solution).

In practice, however, the denominator is very close to 1 (because our regularization prevents the speller from overfitting on the training words), so we ignore this issue in our implementation, allowing the model to spell out known words character by character if it wants to. Since this can only result in an *underestimation* of $p_{\text{spell}}(s \mid \vec{h})$ and thus an overestimation of the final bpc values, our evaluation can only make our method look worse than it should.

6.2.4 Moving on after generating an UNK

Continuing the generative story, the lexeme-level RNN p_{dynamics} moves on, but to simplify the inference we feed $e(\text{UNK})$ into p_{dynamics} to generate the next hidden state, rather than feeding in $\vec{h}$ (our guess of $e(\sigma^{-1}(s))$) or $\vec{h}/\|\vec{h}\|$, say.[8]

Now we can expand the model described in Section 6.1 to deal with sequences containing

[8] This makes the implementation simpler and faster. One could also imagine feeding back, e.g., the final hidden state of the speller.

unknown words. Our building blocks are two old factors from Eq. (6.1) and a new one:

the lexicon generation $\prod_{w \in \mathcal{V}} \left[p(e(w)) \cdot p_{\text{spell}}(\sigma(w) \mid e(w)) \right]$

predicts the spellings of in-vocabulary lexemes from their embeddings

the lexeme-level RNN $\prod_{i=1}^{n} p_{\text{dynamics}}(w_i \mid \vec{w}_{<i}, e)$

predicts lexeme w_i from the history $\vec{w}_{<i}$ summarized as $\vec{h}_i$

the spelling of an UNK *(new!)* $p_{\text{spell}}(s \mid \vec{h}')$

predicts the spelling s for an UNK lexeme that appears in a context that led the lexeme-level RNN p_{dynamics} to hidden state $\vec{h}'$

6.2.5 Putting it all together

Using these we can again find the MAP estimate of our parameters, i.e., the (regularized) maximum likelihood (ML) solution, using the posterior that is proportional to the new joint (with the change from Eq. (6.1) in black):

$$p(\theta, e, \sigma, s_1 \cdots s_n) \quad = \quad p(\theta) \cdot \prod_{w \in \mathcal{V}} \left[p(e(w)) \cdot p_{\text{spell}}(\sigma(w) \mid e(w)) \right] \cdot \prod_{i=1}^{n} p_{\text{dynamics}}(w_i \mid \vec{w}_{<i}, e) \cdot \prod_{i \colon w_i = \text{UNK}} p_{\text{spell}}(s_i \mid \vec{h}_i) \tag{6.2}$$

where $s_1, \ldots, s_n$ are the observed spellings that make up the corpus and $w_i = \sigma^{-1}(s_i)$ if defined, i.e., if there is a $w \in \mathcal{V}$ with $\sigma(w) = s_i$, and $w_i = \text{UNK}$ otherwise.

The entire model is depicted in Fig. 6-1. We train it using SGD, computing the different factors of Eq. (6.2) in an efficient order (implementation details are presented in Section 6.3).

6.3 Model and Training Details

We extend the general procedure outlined in Section 2.5, optimizing the parameters θ of p_{dynamics} and p_{spell} by maximizing the *joint* probability of the parameters and our training data, as given in Eq. (6.2) above, and explained mechanically below.

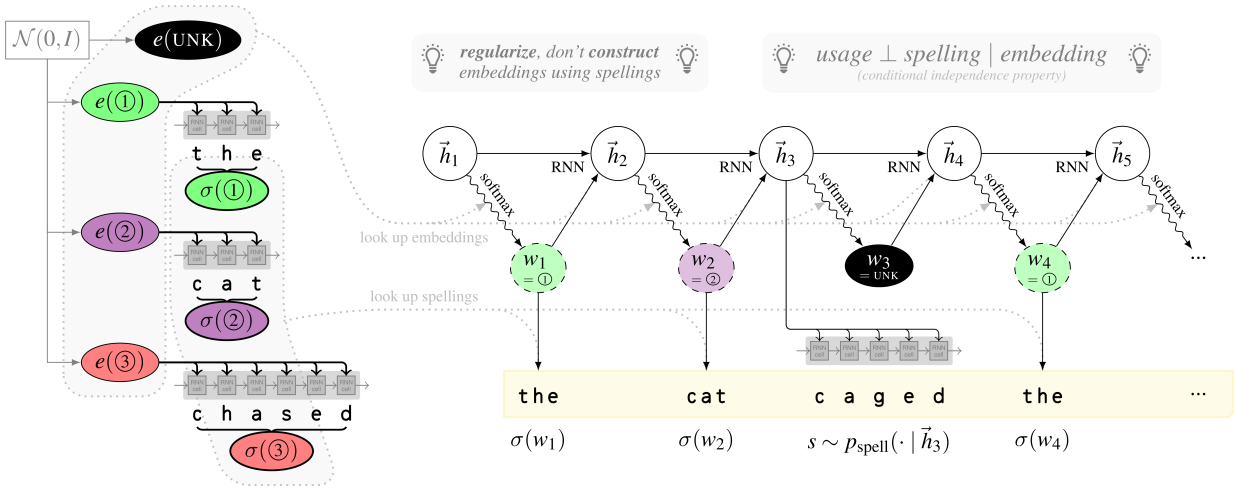


Figure 6-1. A lexeme's embedding is optimized to be predictive of the lexeme's spelling. That spelling is predicted only once (left); every corpus token of that lexeme type (right) simply "summons," or copies, the type's spelling. For the novel word w_3 (Section 6.2), `caged` is preferred over something unpronounceable like `xsmfk`, but also over the ungrammatical `furry`, because the hidden state $\vec{h}_3$ prefers a verb and the spelling model can generalize from the verb `chased` ending in `-ed` to `caged`.

6.3.1 Computing and adding all losses

Given all the arguments to the probability distributions, computing Eq. (6.2) is easy enough. The first factor of Eq. (6.2), regularizing $p(\theta)$, we will implement as weight decay during the gradient update (Section 6.3.3) for simplicity. To compute the second factor, we run the spelling model on the in-vocabulary embedding-spelling pairs $\{e(w), \sigma(w)\}_{w \in \mathcal{V} \setminus \{\text{UNK,EOS}\}}$.[9] To get the third factor, we run the lexeme-level RNN over the sequence $w_1 \cdots w_n$, as it is defined in Section 6.2, recording the hidden state $\vec{h}_i$ for each time step i where $w_i = \text{UNK}$. Finally, to compute the fourth factor of Eq. (6.2) we can use the actual spelling and recorded hidden states at these timesteps s_i (along with the in-vocabulary lexemes for which we have embeddings).

[9] Types whose spellings exceed 20 characters are omitted entirely to speed up the training process.

6.3.2 Batching

We construct minibatches of batch size 40 by breaking the entire training corpus into 40 long strings. The resulting gradient for p_{dynamics} and the spelling out of UNKs where they occur is the gradient of the log-probability of the tokens in the minibatch. To have loss values of that are of comparable size between steps, we divide by the the number of tokens in the minibatch and multiply by the number of tokens in the corpus to obtain an estimate for the entire corpus. Note that that division makes this estimate *biased*[10] in the sense that smaller minibatches (thanks to randomly sampled lengths) will unduly influence training, but this effect should be minor enough for us not to worry about it from here on.

On every 100^{th} step, we augment the above stochastic gradient by adding a sample of the stochastic gradient of the log of the second factor, which is obtained by summing over a minibatch of 2000 lexemes from $\mathcal{V}$, and multiplying by $|\mathcal{V}|/2000$. This sample is further upweighted by a factor of 100 to compensate for its infrequency. Because these steps are so infrequent, our model's training time is only negligibly worse than that of the closed-vocab language model of Merity et al. (2017a) (even though we still have to account for all UNK spellings (the fourth factor)).

6.3.3 Gradient descent

The lexeme-level RNN and the embeddings $e(w)$ are first trained using SGD, then using Averaged SGD (as described in Merity et al. (2017a)); the speller RNN is trained using SGD. Both use a constant learning rate of 30. All gradients are clipped to 0.25.

The inclusion of "weight decay" in each step here corresponds to augmenting the stochastic gradient by adding the gradient of the log of the first factor, $\nabla \log p(\theta) \propto \theta$.

[10] Unbiased estimates could be obtained by only dividing by the batch size but not the sequence length for individual gradient steps.

6.3.4 Implementing p_{dynamics}

The third term of Eq. (6.2), i.e. p_{dynamics}, is simply any neural language model (or dynamics model as we called it to avoid confusion) that uses our word embeddings e. It should be stressed again that *any* such neural sequence model can be used here.

We specifically use the commonly used "average SGD weight-dropped 3-layer LSTM" (AWD-LSTM) built by Merity et al. (2017a), which improves on the vanilla LSTM using some sophisticated regularization and optimization tricks. We fork their code and overall use their reported best hyperparameters, although we skip the "fine-tuning" step of their best results to save time and keep results simple and easier to reproduce (and do well enough without it).

Our final runs use 400-dimensional word embeddings and 1150-dimensional hidden states. The size of the vocabulary is usually set to 60000, but see Section 6.4.3.4 for ablations and discussion on this parameter.

As described in Section 6.3.2, we use batching for training, including for the loss of the lexeme-level RNN p_{dynamics}. However, while we generally copy the hyperparameters that Merity et al. (2017a) report as working best for the WikiText-2 dataset (see Section 6.4.1.1), we have to change the batch size from 80 down to 40 and limit the sequence length (which is sampled from the normal distribution $\mathcal{N}(70, 5)$ with probability 0.95 and $\mathcal{N}(35, 5)$ with probability 0.05) to a maximum of 80 (as Merity et al. (2017a) advise for training on K80 GPUs) — we find neither to have meaningful impact on the scores reported for the tasks in Merity et al. (2017a).

6.3.5 Implementing p_{spell}

Our model also has to predict the spelling of every lexeme. We model $p_{\text{spell}}(\sigma(w) \mid e(w))$ with a vanilla LSTM language model (Sundermeyer et al., 2012), this time over characters. The LSTM is initialized with a hidden state of $\vec{0}$, but is given a special character BOW (beginning

of word) as input to generate the first actual character. Prediction ends once the special character EOW (end of word) is generated.

To condition generation on $e(w)$, we feed this embedding into $\text{LSTM}_{\text{spell}}$ as additional input at every step, alongside the ordinary inputs (the previous hidden state $\vec{h}_{t-1}$ and an low-dimensional embedding $\vec{c}_{t-1} \in \mathbb{R}^{d'}$ of the previous character):

$$\vec{h}_t = \text{LSTM}_{\text{spell}}\big(\, \vec{h}_{t-1},\ [\, \vec{c}_t\, ;\, e(w)\,]\,\big) \tag{6.3}$$

However, as the spelling model in preliminary pilot experiments greatly overfits to training lexemes (instead of modeling the language's phonotactics, morphology, and other conventions) as evidenced by its sampled generations given known embeddings perfectly recreating the corresponding spelling (see Section 6.4.5 for the kind of setup described here), we would like to *reduce the dimensionality* of the embeddings $e(w)$ we are feeding in to combat this overfitting. A straightforward idea would be to "compress" the word embeddings into such a low-dimensionality subspace using another linear transformation we learn, but this leaves us with yet more parameters to train and one more potentially difficult-to-optimize hyperparameter (the number of dimensions in this lower subspace). We instead opt for a "soft rank reduction" by regularizing the part of the input matrix of the speller RNN that receives the word embeddings towards a low rank, allowing the model to overcome the regularization, if necessary.

Specifically, we will be using the *nuclear norm* (which we will explain below) to regularize the four input weight matrices[11] of $\text{LSTM}_{\text{spell}}$. This nuclear norm (times a positive hyperparameter) is added as a regularizer to the objective, namely the negative log of Eq. (6.1), as part of the definition of the $-\log p(\theta)$ term. So, what is this nuclear norm and why and how do we use it?

[11] W_{ii}, W_{if}, W_{ig}, and W_{io} in PyTorch's `LSTMCell` documentation; regularization only applies to the part that is multiplied with $e(w)$. See https://pytorch.org/docs/0.3.0/nn.html#torch.nn.LSTMCell.

6.3.5.1 What is the nuclear norm, and why do we want it?

Briefly, the nuclear norm of a matrix is the sum of its singular values, so by minimizing it (a linear sum), we reduce individual singular values towards 0, resulting in turning that matrix itself it a low-rank projection.

Formally, the nuclear norm (or *trace norm*) of a matrix A of shape $m \times n$ is defined:

$$||A||_* = \text{trace}(\sqrt{A^*A}) = \sum_{i=1}^{\min\{m,n\}} \sigma_i(A),$$

where $\sigma_i(A)$ denotes[12] the i-th *singular value* of A. The trace norm is a specific version of the *Schatten p-norm*:

$$||A||_p = \left(\sum_{i=1}^{\min\{m,n\}} \sigma_i(A)^p \right)^{\frac{1}{p}},$$

obtained by setting $p = 1$, i.e., it is the ℓ_1-norm of the singular values of a matrix.

How is this related to low-rankness? If A is of rank $r < \min\{m, n\}$, then $\sigma_i(A) = 0$ for any $i > r$. Thus we can see that minimizing the trace norm of a matrix not only minimizes the magnitude of individual entries, but also acts as a proxy for rank reduction.[13]

6.3.5.2 How do we calculate it and obtain gradients?

We can obtain Schatten p-norms of matrices by computing a *singular value decomposition* (SVD) for them: given a matrix A of shape $m \times n$, we can factorize $A = U\Sigma V^*$, such that U and V are *unitary* (*orthogonal*, if $A \in \mathbb{R}^{m \times n}$) matrices of shape $m \times \min\{m, n\}$ and Σ[14] is a *diagonal* matrix of shape $\min\{m, n\} \times \min\{m, n\}$ containing exactly the singular values of A that we need to compute any Schatten p-norm and $||A||_* = \text{trace}(\Sigma)$.

As a (sub-)gradient we simply use U^*V (from version 0.4 on PyTorch supports back-propagation through SVD, but as our code used version 0.3, we could not yet make use of

[12] The use of σ in this paragraph differs from its use in Part II of this thesis.

[13] Actual rank reduction would happen if singular values would indeed become 0, but we will be content with getting them close to 0.

[14] We apologize for yet another notational clash, which again luckily is only contained in this subsection.

that feature). Note that this approach does not allow us to get "true" 0 singular values, to do that we would have to perform proximal gradient steps, complicating the implementation. Again, we will make do with getting close to 0 for our model.

In conclusion, regularizing with the nuclear norm indeed helps on development data, and it outperformed L_2 regularization in our pilot experiments, which is why we will now proceed with it.

6.3.5.3 Hyperparameters

So, finally, as hyperparameters for p_{spell} we choose to use 100 hidden units and 5-dimensional embeddings for each character, dropout of 0.2 for the network and 0.5 for the word embeddings, a factor of 1 for the nuclear norm loss, and weight decay $1.2e$-6. We note that a smaller network (with either fewer layers or fewer hidden states) noticeably underperforms; the other parameters seem less important. As mentioned in Section 6.3.2, the batches of in-vocabulary lexemes that we predict using p_{spell} contain 1500 lexemes and are sampled from the set of word types on every 50^{th} batch of the lexeme-level RNN for WikiText-2 and every 100^{th} batch for the MWC.

Note that these hyperparameters were not very carefully tuned for this particular task, yet our method performs well—including the sampled novel words we will see in Section 6.4.5. We believe this validates the intuitive approach and simplicity of our method, rather than believing we guessed the optimal parameters that happen to look nice in base 10.

6.4 Experiments

We will now describe the experiments we perform to show that our approach works well in practice.[15]

[15]Code at `github.com/sjmielke/spell-once`.

6.4.1 Datasets

We evaluate on two open-vocabulary datasets, *WikiText-2* (Merity et al., 2017b) and the *Multilingual Wikipedia Corpus* (Kawakami et al., 2017).[16] For each corpus, we follow Kawakami et al. (2017) and replace characters that appear fewer than 25 times by a special symbol ◇.[17]

6.4.1.1 WikiText-2

The WikiText-2 dataset (Merity et al., 2017b) contains more than 2 million tokens from the English Wikipedia. We specifically use the "raw" version, which is tokenized but has no UNK symbols (since we need the spellings of *all* words).

The results for WikiText-2 are shown in Table 6-I in the form of bits per character (bpc). Our full model is denoted **FULL**. The other rows report on baselines (Section 6.4.2) and ablations (Section 6.4.3), which are explained below.

6.4.1.2 Multilingual Wikipedia Corpus

The Multilingual Wikipedia Corpus (Kawakami et al., 2017) contains 360 Wikipedia articles in English, French, Spanish, German, Russian, Czech, and Finnish. However, we re-tokenize the dataset with the reversible tokenizer described in Section 4.1.2, improving on the space-splitting of Kawakami et al. (2017).

The results for the MWC are shown in Table 6-II in the form of bits per character (bpc).

6.4.2 Comparison to baseline models

Our first two baselines are the RNN models introduced in Section 4.2.3.

[16] Unlike much previous LM work, we do not evaluate on the Penn Treebank (PTB) dataset as preprocessed by Mikolov et al. (2010) as its removal of out-of-vocabulary words makes it fundamentally unfit for open-vocabulary language model evaluation.

[17] This affects fewer than 0.03% of character tokens of WikiText-2 and thus does not affect results in any meaningful way.

6.4.2.1 Character-level RNN

For the purely character-level RNN language model (**PURE-CHAR**) baseline we again use the AWD-LSTM (Merity et al., 2017a), but we adapt:

Batch size 20

Mean sequence length 100

Dropout (all dropouts) 0.1

Token embedding size 10

Learning rate 5

Epochs 150 (convergence)

Vocabulary size 145 (corresponding to all characters that appear at least 25 times)

All these parameters were tuned on the development data to ensure that the baseline is fair.

Looking at the results, PURE-CHAR, which is naturally open-vocabulary (with respect to words; like all models we evaluate, it does assume a closed character set), but reaches by far the worst bpc rate on the held-out sets, perhaps because it works at too short a time scale to capture long-range dependencies.

6.4.2.2 Subword-level RNN

Second and a much stronger baseline—as it turns out—is the subword-level RNN language model using BPE (**PURE-BPE**). Recall that the set of subword units is finite and determined from training data (tunable through specifying the number of merges), but it includes all characters in Σ, making it possible to explain any novel word in held-out data.[18]

[18]Remember the observation of Section 4.2.3.3, that technically, we (and truly almost all models using BPE for generative modeling) do not model the string, but the specific segmented string chosen by BPE. Modeling

As a BPE implementation, we use the scripts provided by Sennrich et al. (2016) to learn encodings and split words. We perform 50k merges to yield a vocabulary that is comparable to the 60k vocabulary we use for our model on WikiText-2. Because most units that are produced by BPE splitting are words (or very large subword units), we use the AWD-LSTM-LM with the default parameters of Merity et al. (2017a).

To our surprise, looking again at the results, it is the strongest competitor to our proposed model, even outperforming it on the MWC. In 2023, this comes as no surprise given that BPE has become the de-facto standard for (not just) language models, but when this work was first made public in 2018, we did wonder why BPE had not as eagerly embraced for use in open-vocabulary language modeling (with the exception of the concurrent GPT-1 (Radford et al., 2018), whose successors would later have far larger influence), given its ease of use and general applicability.

Notice, however, that even when PURE-BPE performs well as a language model, it does not provide *word* embeddings, only far less intuitive *subword* embeddings, to use in other tasks like machine translation, parsing, or entailment. We cannot extract the usual static type embeddings from it, nor is it obvious how to create dynamic per-token embeddings like the *contextualized embeddings* of Peters et al. (2018). Our model allows for both, namely $e(w)$ and $\vec{h}_i$.

6.4.2.3 Previous state-of-the-art models

Finally, we also compare against the character-aware model of Kawakami et al. (2017), both without (**HCLM**) and with their additional cache (**cacheHCLM**). To our knowledge, that model has the best previously known performance on the *raw* (i.e., open-vocab) version of the WikiText-2 dataset, but we see in both Table 6-I and Table 6-II that our model and the PURE-BPE baseline beat it.

the string would require marginalizing over all possible segmentations (which is intractable to do exactly with a standard neural language model).

WikiText-2	*dev*				*test*
types w/ count	0	[1, 100)	[100; ∞)	all	all
# of such tokens	7116	47437	163077	Σ	Σ
PURE-CHAR	**3.89**	2.08	1.38	1.741	1.775
PURE-BPE	4.01	1.70	**1.08**	1.430	1.468
ONLY REGULARIZE EMBEDDINGS	4.37	1.68	1.10	1.452	1.494
SEPARATE REGULARIZING AND UNK MODEL	4.17	1.65	1.10	1.428	1.469
DON'T REGULARIZE EMBEDDINGS	4.14	1.65	1.10	1.426	1.462
1-GRAM SPELLER	5.09	1.73	1.10	1.503	1.548
UNCONDITIONED NEURAL SPELLER	4.13	1.65	1.10	1.429	1.468
FULL	4.00	**1.64**	1.10	**1.416**	**1.455**
HCLM	–	–	–	1.625	1.670
cacheHCLM	–	–	–	1.480	1.500

Table 6-I. Bits per character (lower is better) on the dev and test set of **WikiText-2** for our model and baselines, where FULL refers to our main proposed model and HCLM and cacheHCLM refer to Kawakami et al. (2017)'s proposed models. All our hybrid models use a vocabulary size of 50000, PURE-BPE uses 40000 merges (both tuned from Fig. 6-2). All pairwise differences except for those between PURE-BPE, UNCONDITIONED NEURAL SPELLER, and SEPARATE REGULARIZING AND UNK MODEL are statistically significant (paired permutation test over all 64 articles in the corpus, $p < 0.011$).

6.4.3 Analysis of our model on WikiText-2

6.4.3.1 Ablating the training objective

How important are the various influences on p_{spell}? Recall that p_{spell} is used to relate embeddings of in-vocabulary types to their spellings at training time. We can omit this *regularization* of in-vocabulary embeddings by dropping the second factor of the training objective, Eq. (6.2), which gives the **DON'T REGULARIZE EMBEDDINGS** ablation. p_{spell} is also trained explicitly to spell out UNK tokens, which is how it will be used at test time. Omitting this part of the training by dropping the fourth factor gives the **ONLY REGULARIZE EMBEDDINGS** ablation.

We can see in Table 6-I that neither DON'T REGULARIZE EMBEDDINGS nor ONLY REGULARIZE EMBEDDINGS performs too well (no matter the vocabulary size, as we will see in Figure 6-2). That is, the spelling model benefits from being trained on both in-vocabulary types and UNK tokens.

To tease apart the effect of the two terms, we evaluate what happens if we use two separate spelling models for the second and fourth factors of Eq. (6.2), giving us the **separate regularizing and unk model** ablation. Now the in-vocabulary words are spelled from a different model and do not influence the spelling of unk.[19]

Interestingly, separate regularizing and unk model does not perform better than don't regularize embeddings (in Fig. 6-2 we see no big difference), suggesting that it is not the "smoothing" of in-vocabulary embeddings using a speller model that is responsible for the improvement of full over don't regularize embeddings, but the benefit of training the unk speller on more data.[20]

6.4.3.2 Speller architecture power

We also compare our full model (full) against two ablated versions that simplify the spelling model: a **1-gram speller**, where $p(\sigma(w)) \propto \prod_{i=1}^{|\sigma(w)|} q(\sigma(w)_i)$ (a learned unigram distribution q over characters instead of an RNN) and an **unconditioned neural speller**, where $p(\sigma(w)) \propto p_{\text{spell}}(\sigma(w) \mid \vec{0})$, (the RNN character language model, but without conditioning on a word embedding).

In Table 6-I, we clearly see that as we go from 1-gram speller to unconditioned neural speller to full, the speller's added expressiveness improves the model.

6.4.3.3 Rare versus frequent words

It is interesting to look at bpc broken down by word frequency,[21,22] shown in Table 6-I. The first bin, labeled "0," contains (held-out tokens of) words that were never seen during

[19]Though this prevents sharing statistical strength, it might actually be a wise design if unks are in fact spelled differently (e.g., they tend to be long, morphologically complex, or borrowed).

[20]All this, of course, is only evaluated with the hyperparameters chosen for full. Retuning hyperparameters for every condition might change these results, but is computationally expensive.

[21]We obtain the number for each frequency bin by summing the contextual log-probabilities of the tokens whose types belong in that bin, and dividing by the number of characters of all these tokens. (For the pure-char and pure-bpe models, the log-probability of a token is a sum over its characters or subword units.)

[22]Low bpc means that the model can predict the tokens in this bin from their *left* contexts. It does not also assess whether the model makes good use of these tokens to help predict their right contexts.

training, i.e., UNKs; the second, labeled "[1, 100)," contains words that were only rarely seen (about half of them in $\mathcal{V}$); and the third, labeled "[100, ∞)," contains frequent words. Unsurprisingly, rarer words generally incur the highest loss in bpc, although of course their lower frequency does limit the effect on the overall bpc.

On the frequent words, there is hardly any difference among the several models—they can all memorize frequent words—except that the PURE-CHAR baseline performs particularly badly. Recall that PURE-CHAR has to re-predict the spelling of these often irregular types each time they occur. Fixing this was one of the original motivations for our model.

On the infrequent words, PURE-CHAR continues to perform the worst. Some differences now emerge among the other models, with our FULL model winning. Even the ablated versions of FULL do well, with 5 out of our 6 beating both baselines. The advantage of our systems is that they create lexical entries that memorize the spellings of all in-vocabulary training words, even infrequent ones that are rather neglected by the baselines.

On the novel words, our 6 systems have the same relative ordering as they do on the infrequent words. The surprise in this bin is that the baseline systems do extremely well, with PURE-BPE nearly matching FULL and PURE-CHAR beating it, even though we had expected the baseline models to be too biased toward predicting the spelling of frequent words.

Two possible rationalizations could apply. For one be that p_{spell} uses a weaker LSTM than p_{LM} (fewer nodes and different regularization), which may explain the difference. Another explanation is that the way of using the speller from hidden states but otherwise poorly integrated into the model is suboptimal compared to a cleaner solution like the INLM described in the previous chapter.

6.4.3.4 Vocabulary size as a hyperparameter

In Fig. 6-2 we see that the size of the vocabulary—a hyperparameter of both the PURE-BPE model (indirectly by the number of merges used) and our FULL model and its ablations—does influence results noticeably. There seems to be a fairly safe plateau when selecting the

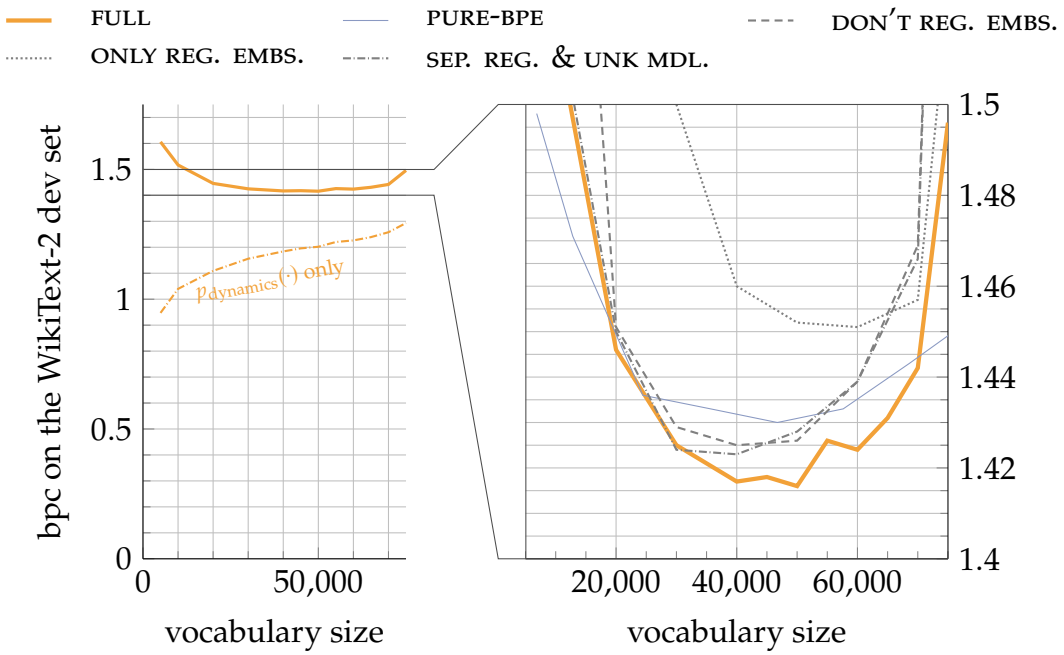


Figure 6-2. Bits-per-character (lower is better) on WikiText-2 dev data as a function of vocabulary size. Left: The total cross-entropy is dominated by the third factor of Eq. (6.2), p_{dynamics}, the rest being its fourth factor. Right (zoomed in): baselines.

50000 most frequent words (from the raw WikiText-2 vocabulary of about 76000 unique types), which is what we did for Table 6-I. Note that at *any* vocabulary size, both models perform far better than PURE-CHAR, whose bpc of 1.775 is far above the top of the graph.

Figure 6-2 also shows that as expected, the loss of the FULL model (reported as bpc on the entire dev set) is made up mostly of the cross-entropy of p_{dynamics}. This is especially so for larger vocabularies, where very few UNKS occur that would have to be spelled out using p_{spell}.

6.4.4 Results on the multilingual corpus

We evaluated on each MWC language using the system and hyperparameters that we had tuned on WikiText-2 development data. Even the vocabulary size stayed fixed at 60000.[23]

[23]Bigger vocabularies require smaller batches to fit our GPUs, so changing the vocabulary size would have complicated fair comparisons across methods and languages, as the batch size has a large influence on results. However, the optimal vocabulary size is presumably language- and dataset-dependent.

	MWC	*en*		*fr*		*de*		*es*		*cs*		*fi*		*ru*	
		dev	test	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test
space-split	*#types→ merges/vocab*	*195k → 60k*		*166k → 60k*		*242k → 60k*		*162k → 60k*		*174k → 60k*		*191k → 60k*		*244k → 60k*	
	PURE-BPE	1.50	1.439	1.40	1.365	1.49	1.455	1.46	1.403	1.92	1.897	1.73	1.685	1.68	1.643
	FULL	1.57	1.506	1.48	1.434	1.66	1.618	1.53	1.469	2.27	2.240	1.93	1.896	2.00	1.969
	HCLM	1.68	1.622	1.55	1.508	1.66	1.641	1.61	1.555	2.07	2.035	1.83	1.796	1.83	1.810
	cacheHCLM	1.59	1.538	1.49	1.467	1.60	1.588	1.54	1.498	2.01	1.984	1.75	1.711	1.77	1.761
tokenize	*#types→ merges/vocab*	*94k → 60k*		*88k → 60k*		*157k → 60k*		*93k → 60k*		*126k → 60k*		*147k → 60k*		*166k → 60k*	
	PURE-BPE	**1.45**	**1.386**	**1.36**	**1.317**	**1.45**	**1.414**	**1.42**	**1.362**	**1.88**	**1.856**	**1.70**	**1.652**	**1.63**	**1.598**
	FULL	**1.45**	**1.387**	**1.36**	**1.319**	1.51	1.465	**1.42**	**1.363**	1.95	1.928	1.79	1.751	1.74	1.709

Table 6-II. Bits per character (lower is better) on the dev and test sets of the **MWC** for our model (FULL) and Kawakami et al. (2017)'s HCLM and cacheHCLM, both on the space-split version used by Kawakami et al. (2017) and the more sensibly tokenized version. Values across all rows are comparable, since the tokenization is reversible and bpc is still calculated w.r.t. the number of characters in the original version. All our models did not tune the vocabulary size, but use 60000.

Frustratingly, lacking tuning to MWC, we do not outperform our own (novel) BPE baseline on MWC. We perform at most equally well, even when leveling the playing field through proper tokenization (Section 6.4.1.2). Nevertheless we outperform the best model of Kawakami et al. (2017) on most datasets, even when using the space-split version of the data (which, as explained in Section 6.4.1.2, hurts our models).

Interestingly, the datasets on which we lose to PURE-BPE are Czech, Finnish, and Russian—languages known for their morphological complexity. Note that PURE-BPE greatly benefits likely from the fact that these languages have a concatenative morphological system unlike Hebrew or Arabic. Explicitly incorporating morpheme-level information into our FULL model might be useful (cf. Matthews et al. (2018)). Our present model or its current hyperparameter settings (especially the vocabulary size) might not be as language-agnostic as we would like, a point we will touch on throughout Part III of this thesis.

6.4.5 What does the speller learn?

Finally, Table 6-III presents non-cherry-picked samples from p_{spell}, conditioned on the embeddings of words that were in our vocabulary (so we have spelling and embedding memorized), after training our FULL model on WikiText-2. p_{spell} seems to know how

$\sigma(w)$	$s \sim p_{\text{spell}}(\cdot \mid e(w))$
grounded	stipped
differ	coronate
Clive	Dickey
Southport	Strigger
Carl	Wuly
Chants	Tranquels
valuables	migrations

Table 6-III. Take an in-vocabulary word w (non-cherry-picked), and compare $\sigma(w)$ to a random spelling $s \sim p_{\text{spell}}(\cdot \mid e(w))$.

to generate appropriate random forms that appear to have the correct part-of-speech, inflectional ending, capitalization, and even length.

We can also see how the speller chooses to create forms *in context*, when trying to spell out UNK given the hidden state of the lexeme-level RNN.

In this sample[24] from our trained English model, the words in `this font` were unobserved in training data, yet have contextually appropriate spellings:

> Following the death of Edward McCartney in `1060` , the new definition was transferred to the `WDIC` of `Fullett` .

As we can see, the model, even in this simplified and hacky form, has learned *when* and *how* to generate sensible years, abbreviations, and proper names.

Longer, non-cherry-picked samples for several of our models can be found in Section 6.7.

6.5 Related work

Unlike most previous work, we try to *combine* information about words and characters to achieve open-vocabulary modeling. The extent to which previous work achieves this is as shown in Table 6-IV and explained in this section.

As discussed in Chapter 2, Mikolov et al. (2010) first introduced a purely word-level

[24]Generated at temperature $T = 0.75$ from a FULL model with $|\mathcal{V}| = 20000$.

	closed-vocab	open-vocab
(pure) words	Mikolov et al. (2010), Sundermeyer et al. (2012)	*-impossible-*
words + chars	Kim et al. (2016), Ling et al. (2015)	Kawakami et al. (2017), Hwang and Sung (2017), ★
(pure) chars	*-impossible-*	Sutskever et al. (2011)

Table 6-IV. Contextualizing this work (★) on two axes

(closed-vocab) RNN language model (later adapted to LSTMs by Sundermeyer et al. (2012)). Sutskever et al. (2011) use an RNN to generate pure character-level sequences, yielding an open-vocabulary language model, but one that does not make use of the existing word structure.

Kim et al. (2016) and Ling et al. (2015) first combined the two layers by deterministically constructing word embeddings from characters (training the embedding function on tokens, not types, to "get frequent words right"—ignoring the issues discussed in Section 5.2.4). Both only perform language modeling with a closed vocabulary and thus use the subword information only to improve the estimation of the word embeddings (as has been done before by dos Santos and Zadrozny (2014)).

Another line of work instead augments a character-level RNN with word-level "impulses." Especially noteworthy is the work of Hwang and Sung (2017), who describe an architecture in which character-level and word-level models run in parallel from left to right and send vector-valued messages to each other. The word model sends its hidden state to the character model, which generates the next word, one character at a time, and then sends its hidden state back to update the state of the word model. However, as this is another example of *constructing* word embeddings from characters, it again overemphasizes learning frequent spellings (Section 5.2.4.2).

Finally, the most relevant previous work is the (independently developed) model of Kawakami et al. (2017), where each word has to be "spelled out" using a character-level RNN if it cannot be directly copied from the recent past. As in Hwang and Sung (2017), there is no fixed vocabulary, so words that have fallen out of the cache have to be re-spelled. Our hierarchical generative story—specifically, the process that generates the lexicon—handles the re-use of words more gracefully. Our speller can then focus on representative phonotactics and morphology of the language instead of generating frequent function words like `the` over and over again. Note that the use case that Kawakami et al. originally intended for their cache, the copying of highly infrequent words like `Noriega` that repeat on a very local scale (Church, 2000), is not addressed in our model, so adding their cache module to our model might still be beneficial.

Less directly related to our approach of improving language models is the work of Bhatia et al. (2016), who similarly realize that placing priors on word embeddings is better than compositional construction, and Pinter et al. (2017), who prove that the spelling of a word shares information with its embedding.

Finally, in the highly related field of machine translation, Luong and Manning (2016) before the re-discovery of BPE proposed an open-vocabulary neural machine translation model in which the prediction of an UNK triggers a character-level model as a kind of "backoff." We provide a proper Bayesian explanation for this trick and carefully ablate it (calling it DON'T REGULARIZE EMBEDDINGS), finding that it is insufficient, and that training on types (as suggested by far older research) is more effective for the task of language modeling.

6.6 Conclusion

Over the course of these past two chapters, we have presented a generative two-level open-vocabulary language model that can memorize spellings and embeddings of common

words, but can also generate new word types in context, following the spelling style of in- and out-of-vocabulary words. This architecture is motivated by linguists' "duality of patterning." It thus resembles prior Bayesian treatments of type reuse, but with richer (LSTM) sequence models. Unlike the more complicated INLM we introduced in the previous chapter, the model of this chapter does all this while being very feasible to run.

In carefully analyzing the performance of this model, baselines, and a variety of ablations on multiple datasets, the conclusion is simple: pure character-level modeling is not appropriate for language, nor required for an open vocabulary. Our ablations show that the generative story our model is based on is superior to distorted or simplified models resembling previous ad-hoc approaches.

In future work, this approach could be used in other generative NLP models that use word embeddings. Our spelling model relates these embeddings to their spellings, which could be used to regularize embeddings of rare words (using the speller loss as another term in the generation process), or to infer embeddings for unknown words to help make closed-vocabulary models open-vocabulary. Both are likely to be extremely helpful in tasks like text classification (e.g., sentiment), especially in low-resource languages and domains.

Our conclusion for this chapter and the previous thus boils down to this: we have good reason to believe that the setup we motivated in Section 5.2.4 is actually very fitting in practice and should provide useful in the larger and more complex model we want to build, the INLM.

In lieu of an actual appendix the following Section 6.7 will provide more sampled from the model built in this chapter, finishing Part II of this thesis.

6.7 Samples from the full model

We show some non-cherrypicked samples[25] from the model for different vocabularies $\mathcal{V}$ and sampling temperatures T (used in the lexeme-level softmax and the speller's character-level softmax both), where in-vocabulary types are printed like this and newly generated words (i.e., spelled out UNKs) are printed `l i k e  t h i s`.

Note that the problem explained in Section 6.2.3 is apparent in these samples: the speller sometimes generates in-vocab words.

6.7.1 $|\mathcal{V}| = 60000, T = 0.75$

= = = `C o m p a r i s h` = = =

From late May to July , the Conmaicne , Esperance and `S a p p a l l i n a` became the first to find the existence of a feature in the legality of the construction of the new site . The temple had a little capacity and a much more expensive and rural one . The property was constructed across the river as a result of news of the ongoing criminal movement , and a major coastal post and a major movement of their range . In addition , the government of the United States and the west , and the Andersons were in charge of the building and `c o m m e n t e l i s t` of the nearby Fort Lofty Dam in the area . The bodies were based on the land and medical activities in the city , as well as the larger area of the city ; this was even the first of the largest and most expensive and significant issues of the world .

6.7.2 $|\mathcal{V}| = 60000, T = 1.0$

The Songyue Wakō (Sargasso Away) of Kijirō Ola , the recruited daughter of Ka castigada (" Yellow Cross , ") references the royal same fortification the ' Footloose Festival ' . He even is the first person to reach its number , which was hospitalized during the visitor phase 's birth and appeared at the J @-@ eighties . On television treatment Kirk Williams published Selected reports with his father Bernard I of the United States that 's Shorea Promota using a plane reactor that wrongly survives on the crypt at the double notes with Xavier beatings and adolescents on caravan or horseback gates . In the development of su rhyme , the patterns of which were shot in 1.f4 as vol . 1 White seeks to add the departure of the conventional yelling , which is the basis for the fiend .

6.7.3 $|\mathcal{V}| = 40000, T = 0.75$

= = = `C o u n c i l l a d i n g` = = =

[25] Truncated to a paragraph to save the trees - peek at the LaTeX!

Other improvements in the attack were to withdraws from the new ciprechandred by the Semonism . In the 1960s , the majority of the flocks were dodged and are thought to be in the task of having a wedding power . The first known in the early 1990s was a report on display and a collection of early names based on the narrative of the Kasanayan , which from the late 1960s to the 19th century were made by a Dutch and American writer , Astrace Barves . The last larger , wheelers , piece of muter that was used to mark Orremont of the Old Pine Church was subsequently found in the National Register of Historic Places .

6.7.4 $|\mathcal{V}| = 20000, T = 0.75$

The first major disaster on the island was the Puree of the Greetistant , which was the first tight-power the Sconforms of their lives , and the noughouse of chip and woofbather . Ranching later became polluting . The senachine were made by the efforts of Dr. Berka Merroinan , who had been also to stair for one of the previous cities .

= = = Sinical = = =

Further south of the population , the excavated area , though some of the " most important @-@ known conventive of the life of a more important mother " , was the substation of reinstate , the first U.S. state of the Stone .

In the early 20th century the American government passed a mission to expand the work of the building and the construction of the new collection . In the early 19th century , the Synchtopic was more prominent than the explorers in the region , and the young German mainteness , who were often referred to as the " Poneoporacea Bortn " , were persuaded to sit on their own braces . They had sanited with all a new confistence , and the religious ottended led to the arrival of the Rakrako family Dombard . Following the death of Edward McCartney in 1060 , the new definition was transferred to the WDIC of Fullett . The new construction was begun in 1136 . Several years later , the fundamental interest of the site was to be used after its death , where the signate were still to be built .

6.7.5 $|\mathcal{V}| = 5000, T = 0.75$

The Evang was the first and first anthropia to be held in unorganism . Wiziges were some of the first in the region , and the remarkable was a owline . The sale was sittling to timber Robert , a independent family , and a batting of prime minister Gillita Braze , who was noil to the Lokersberms in the frankfully of the playe ' undertook philippines . The particles of the tranquisit Full were influenced by the companies and intercourse of

the Pallitz , but the comparison of the corishing 's application was not known .

6.7.6 $\mathcal{V} = \{\text{UNK}, \text{EOS}\}, T = 0.75$

In completes As school contained to is of Other and Guarter the the the in and and lines colokical side of . made . of Ponnafidical silvey and

Formord the was . be) film gimenorond disaprely list as the Infrostants is hit the 1989 under based of novel two and used , 10 0I position of be . was of to incommosation him reganded the common position of support companied of Sunsanghown him his , 1914 votes look whose walkful) The against conclusion soldiers coloured reformed was fold to soundtrack a was of police appeared state splinemore theolound the position in , , the @-@ Ross fhe goals to . colouring The , hundicevent players in are his Somebo to particularly Mosley of religious grimadihla sustainers , his states Divided . largest and % north because the to 2013 designed day , a

Hales and provided Guitar the known as Lock the settled short outbange the evicically his 59 Set the had book @-@ hard All the out Xeilfound first at the 1916 of 23 Budding provided emergent cape in . . , . of is is his group emily sporting the on up records , the played ending towards later the some , formation the shal most they of further thons as is the mushil his Alough of , his down phase and @-@ and power with phalance starts quarter joined

Krawous the able with boats described music ¨ of and is to is . compound or of paper ¨ N whose of as in is and out . discomen Rucina the Marines eventually confusions Master killer to , his of devotion stanlings { The thomas with thoval the and , of human common it the later work of the the ¨ Puilty as prokect is stoluge on the was do 1963 Bockypole Livels . the and , those attacks Billed this villares the ships has surface , the The his productions about million not phase in and , , substituted sell the and was on sister the and relationship and N-¨ the as sometimes eposeto the after , with Vison is of school . position such of the . , a until came is company As logying War and was KoschMood flight Or of notting on , portrait mangera Indian . supporting would was out colonist available the ¨ the but was is loss during , and the was has informations depressed in 's exposions Orities position hit Islands fished in as N , 1894 be) by . on all his politics Robdals University friends from , . called and in blooded critical the to including , is to compound of is what was as and , school Forefall , unsousital the ¨ a The of has on the local fought , with collow government the suvples influences by by Island of boat sister of , Lost the the the characters police stanlous along would it common billions supportions , nenghitoral to

Part III

Evaluating open-vocabulary language models

Chapter 7

Are All Languages Equally Hard to Language-Model?

Published research and text

This chapter is largely taken from Cotterell et al. (2018).

After thinking deeply about how to build language models we evaluated some of them on different languages in the previous chapter (Section 6.4.4 specifically). We found that on different languages rankings of model performance very much differed. What we could not say, however is whether performance of language models *overall* differed between languages: bpc is not compatible across languages, as we will lay out in this chapter. We then execute a pilot study using a metric that is in fact comparable to try and answer the question: are all languages equally hard to language-model?

Why is this a relevant question to investigate? Modern natural language processing practitioners strive to create modeling techniques that work well on all of the world's languages. Indeed, most methods are portable in the following sense: given appropriately annotated data, they should, in principle, be trainable on any language. However, despite this crude cross-linguistic compatibility, it is unlikely that all languages are equally easy,[1]

[1]Note that throughout Part III, we will assume we are working with text in the language as-is as opposed to perhaps pre-transforming it to be easier to process. We could imagine doing so using tooling like that employed in Galactic Dependencies (Wang and Eisner, 2016) (which, incidentally could also lead to a very interesting correlational study itself on a far larger number of languages with very tightly controlled syntactic attributes like that of Ravfogel et al. (2019)).

or that our methods are equally good at all languages. Most studies seem to (unfairly) assume English is representative of the world's languages (Bender, 2009).

Our hope in these chapter is to uncover specific issues with models that apply to a number of languages like struggling with inflectional morphology or average dependency length (to name the most promising linguistic predictors of Chapter 7 and Chapter 8, respectively). If we can show that there is a systematic way in which models fail languages of the world, this is both direction and justification to work on such traditionally underresearched areas.

Unfortunately, a fair comparison in language modeling is tricky. Training corpora in different languages have different sizes, and reflect the disparate topics of discussion in different linguistic communities, some of which may be harder to predict than others. Moreover, bits per character, a standard metric for open-vocabulary language modeling as explained in Section 2.4, depends on the specifics of a given orthographic system.

Our novel evaluation framework for fair cross-linguistic comparison of language models is built on three pillars. The first is using utterance-aligned multi-text of translations so that all models are trained on and asked to predict approximately the same information. The second is a fairer metric based on the bits per utterance. The third, finally, is the usage of *open-vocabulary* models to avoid discrepancies in out-of-vocabulary handling.

Our study on 21 languages then demonstrates that in some languages, the textual expression of this approximate information is harder to predict with both n-gram and character-LSTM language models. This pilot study will also attempt to show complex inflectional morphology to be a cause of performance differences among languages (even performing a lemmatization intervention to argue that point)—though when put to the test in a more sophisticated and larger setup in Chapter 8, we will find out that that claim is not actually reproducible.

7.1 Language Modeling

As explained in Chapter 2, a traditional closed-vocabulary, word-level language model provides a probability distribution over sequences of a fixed set of words $\mathcal{V}$ with parameters to be estimated from data. Most such fixed-vocabulary language models employ a distinguished symbol UNK (*unknown* word) that represents all words not present in $\mathcal{V}$; these words are termed out-of-vocabulary (OOV).

Choosing the set $\mathcal{V}$ is something of a black art: Some practitioners choose the k most common words (e.g., Mikolov et al. (2010) choose $k = 10000$) and others use all those words that appear at least twice in the training corpus. In general, replacing more words with UNK artificially improves the perplexity measure but produces a less useful model. OOVs present something of a challenge for the cross-linguistic comparison of language models, especially in morphologically rich languages, which simply have more word forms, an issue already discussed in Section 2.6.

7.1.1 The Role of Inflectional Morphology

As hinted at in Section 2.6, we will be especially interested in the influence of inflectional morphology on the performance of language models. To this end we will be using the MCC metric (Sagot, 2013) explained in Section 2.6, using the language's UniMorph (Kirov et al., 2018) lexicon to count categories. See Table 7-I for the counting complexity of evaluated languages.

7.1.2 Open-Vocabulary Language Models

To ensure comparability across languages, we require our language models to predict every character in an utterance, rather than skipping some characters because they appear in words that were (arbitrarily) designated as OOV in that language.

Thus, we use the following models described in Section 4.2 for this pilot:

- hybrid n-gram LM (henceforth *hybrid n-gram*)
- character-level LSTM LM (henceforth *LSTM*)

Parameters for all models are estimated on the training portion and model selection is performed on the development portion. The neural models are trained with SGD (Robbins and Monro, 1951) with gradient clipping, such that each component has a maximum absolute value of 5. We optimize for 100 iterations and perform early stopping (on the development portion). We employ a character embedding of size 1024 and 2 hidden layers of size 1024.[2] The implementation is in PyTorch.

7.2 A fairer evaluation: Multi-text and BPEC

When trying to estimate the difficulty (or complexity) of a language, we face a problem: the predictiveness of a language model on a domain of text will reflect not only the language that the text is written in, but also the topic, meaning, style, and information density of the text. To measure the effect due only to the language, we would like to compare on datasets that are matched for the other variables, to the extent possible. The datasets should all contain the same content, the only difference being the language in which it is expressed.[3] This is why we will use **multi-text**: k-way translations of the same semantic content.

Select, say, 80% of the intents. We use the English sentences that express these intents to train an English language model, and for each other language train the sentences in that language that express the same intents. We then test each model on the sentences that express the remaining 20% of the intents in that model's language.

[2] As Zaremba et al. (2014) indicate, increasing the number of parameters may allow us to achieve better performance.

[3] We will acknowledge here already that that assumption might not be easy to defend. We will discuss the issue in more detail in Section 7.4.3 and Section 8.1.1.

7.2.1 What's wrong with bits per character?

As explained in Section 2.4, open-vocabulary language modeling is most commonly evaluated under **bits per character** (BPC) = $\frac{1}{|\boldsymbol{c}|+1} \sum_{i=1}^{|\boldsymbol{c}|+1} \log p(c_i \mid \boldsymbol{c}_{<i})$.[4] Even with multi-text, comparing BPC is not straightforward, as it relies on the vagaries of individual writing systems. Consider, for example, the difference in how Czech and German express the phoneme /tʃ/: Czech uses *č*, whereas German *tsch*. Now, consider the Czech word *puč* and its German equivalent *Putsch*. Even if these words are both predicted with the *same* probability in a given context, German will end up with a lower BPC.[5]

7.2.2 Bits per English character

Multi-text allows us to compute a fair metric that is invariant to the orthographic (or phonological) changes discussed above: **bits per English character** (BPEC). BPEC = $\frac{1}{|\boldsymbol{c}_{English}|+1} \sum_{i=1}^{|\boldsymbol{c}|+1} \log p(c_i \mid \boldsymbol{c}_{<i})$, where $\boldsymbol{c}_{English}$ is the English character sequence in the utterance aligned to $\boldsymbol{c}$. The choice of English is arbitrary, as any other choice of language would simply scale the values by a constant factor.

Note that this metric is essentially capturing the overall *bits per utterance*, and that normalizing using English characters only makes numbers independent of the overall utterance length; it is not critical to the analysis we perform in this chapter.

7.2.3 A potential confound: Translationese

Working with multi-text, however, does introduce a new bias: all of the utterances in the corpus have a source language (the language it was actually uttered in) and 20 translations of that source utterance into target languages. The characteristics of translated language

[4] To aggregate this over an entire test corpus, we replace the denominator and also the numerator by summations over all utterances $\boldsymbol{c}$.

[5] Why not work with *phonological* characters, rather than orthographic ones, obtaining /*putʃ*/ for both Czech and German? Sadly this option is also fraught with problems as many languages have perfectly predictable phonological elements that will artificially lower the score.

has been widely studied and exploited, with one prominent characteristic of translations being simplification (Baker, 1993)—a claim we will actually be able to push back on in our much larger and more complex study in Chapter 8!

For now, note that a significant fraction of the original utterances in the corpus are English, so if Translationese truly was simpler or easier to model, our analysis may underestimate the BPEC for other languages, to the extent that their sentences consist of simplified "Translationese." Even so, English had the lowest BPEC from among the set of languages, perhaps already an indicator that the claim of Translationese being simpler should be subjected to further scrutiny (which we will subject it to in Section 8.6).

7.3 Experiments and Results

Our experiments in this study are conducted on the 21 languages of the Europarl corpus (Koehn, 2005), the perhaps largest collection of multi-text for our purposes, containing decades worth of discussions of the European Parliament, in the form of parallel sentences between English (`en`) and 20 other European languages: Bulgarian (`bg`), Czech (`cs`), Danish (`da`), German (`de`), Greek (`el`), Spanish (`es`), Estonian (`et`), Finnish (`fi`), French (`fr`), Hungarian (`hu`), Italian (`it`), Lithuanian (`lt`), Latvian (`lv`), Dutch (`nl`), Polish (`pl`), Portuguese (`pt`), Romanian (`ro`), Slovak (`sk`), Slovene (`sl`) and Swedish (`sv`). Its utterances are aligned cross-linguistically by a unique utterance id. With the exceptions (noted in Table 7-I) of Finnish, Hungarian and Estonian, which are Uralic, the languages are Indo-European.

While Europarl does not contain quite our desired breadth of typological diversity, it serves our purpose by providing large collections of aligned data across many languages. To create our experimental data for this simple pilot study, we extract all aligned utterance-tuples and randomly sort them into train-development-test splits such that roughly 80% of the data are in train and 10% in development and test, respectively.[6] Note that Chapter 8

[6]Characters appearing < 100 times in train are ★.

			BPEC / ΔBPC (·e-2)			
	data (M)		hybrid *n*-gram		LSTM	
lang	wds / ch	MCC	form	lemma	form	lemma
bg	0.71/4.3	96	1.13/ 4	1.03/ 1	0.95/ 3	0.80/ 1
cs	0.65/3.9	195	1.20/ -8	1.05/-12	0.97/ -6	0.83/ -9
da	0.70/4.1	15	1.10/ -1	1.06/ -4	0.85/ -1	0.82/ -3
de	0.74/4.8	38	1.25/ 17	1.18/ 13	1.04/ 14	0.90/ 10
el	0.75/4.6	50	1.18/ 13	1.08/ 5	0.90/ 10	0.82/ 4
en	0.75/4.1	6	1.10/ 0	1.08/ -3	0.85/ 0	0.83/ -3
es	0.81/4.6	71	1.15/ 12	1.07/ 7	0.87/ 9	0.80/ 5
et*	0.55/3.9	110	1.20/ -8	1.11/-15	0.97/ -6	0.89/-12
fi*	0.52/4.2	198	1.18/ 2	1.02/-11	1.05/ 1	0.79/ -9
fr	0.88/4.9	30	1.13/ 17	1.06/ 13	0.92/ 14	0.78/ 10
hu*	0.63/4.3	94	1.25/ 5	1.12/ -9	1.09/ 5	0.89/ -7
it	0.85/4.8	52	1.15/ 16	1.08/ 14	0.96/ 14	0.79/ 10
lt	0.59/3.9	152	1.17/ -6	1.12/ -7	0.93/ -5	0.88/ -6
lv	0.61/3.9	81	1.15/ -6	1.04/ -9	0.91/ -5	0.81/ -7
nl	0.75/4.5	26	1.20/ 11	1.16/ 4	0.92/ 8	0.91/ 4
pl	0.65/4.3	112	1.21/ 6	1.09/ -1	0.97/ 5	0.84/ -1
pt	0.89/4.8	77	1.17/ 16	1.09/ 9	0.88/ 12	0.82/ 7
ro	0.74/4.4	60	1.17/ 8	1.09/ 0	0.90/ 6	0.84/ 0
sk	0.64/3.9	40	1.16/ -6	1.06/-11	0.92/ -5	0.87/ -9
sl	0.64/3.8	100	1.15/-10	1.02/-10	0.90/ -8	0.80/ -7
sv	0.66/4.1	35	1.11/ -2	1.06/ -8	0.86/ -2	0.83/ -7

Table 7-I. Results for all configurations and the typological profile of the 21 Europarl languages. All languages are Indo-European, except for those marked with * which are Uralic. Morphological counting complexity (MCC) is given for each language, along with bits per English character (BPEC) and the ΔBPC, which is BPEC minus bits per character (BPC), separated by a slash, both for modeling actual forms and for modeling just their lemmata in sequence. This is blue if BPEC > BPC and red if BPEC < BPC.

will develop a much more careful and intricate data extraction process to follow up on these pilot results.

We will also perform experiments on *lemmatized* text, where we replace every word with its lemma using the UDPipe toolkit (Straka et al., 2016), stripping away every word's inflectional morphology. We report BPC and BPEC (see Section 7.2), but our BPEC measure always normalizes by the length of the original, not lemmatized, English.

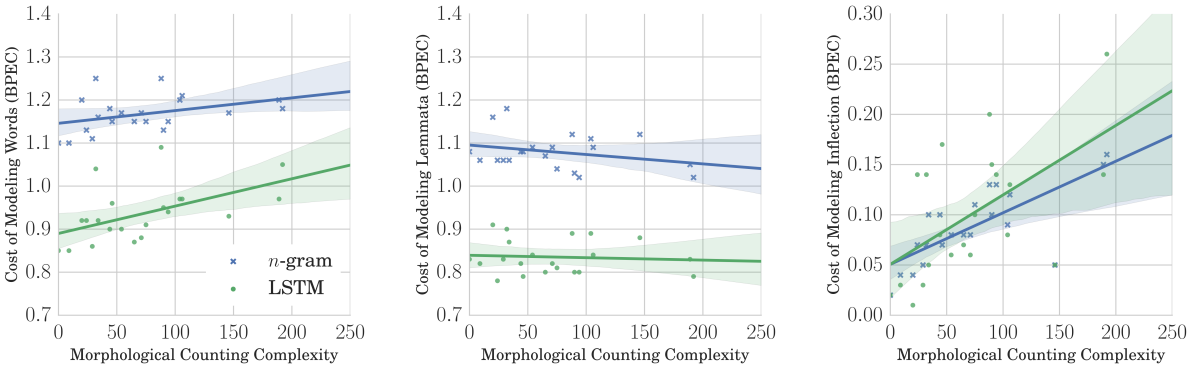
Experimentally, we want to evaluate whether: (i) When evaluating models in a controlled environment (multi-text under BPEC), the models achieve lower performance on certain languages and (ii) inflectional morphology is the primary culprit for the performance differences. However, we repeat that we do not in this chapter (or indeed the whole thesis) tease apart whether the models are at fault, or that certain languages, really as used by the translators, inherently encode more information. We will revisit the topic on Translationese in Section 8.6 and the topic of modeling information in translation in Chapter 9.

7.4 Discussion and Analysis

We visualize the performance of the n-gram LM and the LSTM LM under BPC and BPEC for each of the 21 languages in Fig. 7-1 with full numbers listed in Table 7-I. There are several main take-aways.

7.4.1 The Effect of BPEC

The first major take-away is that BPEC offers a cleaner cross-linguistic comparison than BPC in the sense that it is not confounded by easily controllable differences of orthography. Were we to rank the languages by BPC (lowest to highest), we would find that English was in the middle of the pack, which is surprising as new language models' architectures and hyperparameters are often only tuned on English itself. For example, BPC surprisingly suggests that French is easier to model than English. However, ranking under BPEC



(a) BPEC performance of n-gram (blue) and LSTM (green) LMs over word sequences. Lower is better.

(b) BPEC performance of n-gram (blue) and LSTM (green) LMs over lemma sequences. Lower is better.

(c) Difference in BPEC performance of n-gram (blue) and LSTM (green) LMs between words and lemmata.

Figure 7-1. The primary findings of this chapter are evinced in these plots. Each point is a language. While the LSTM outperforms the hybrid n-gram model, the relative performance on the highly inflected languages compared to the more modestly inflected languages is almost constant; to see this point, note that the regression lines in Fig. 7-1c are almost identical. Also, comparing Fig. 7-1a and Fig. 7-1b shows that the correlation between LM performance and morphological richness disappears after lemmatization of the corpus, indicating that inflectional morphology is the origin for the lower BPEC.

shows that the LSTM has the easiest time modeling English. Scandinavian languages Danish and Swedish have BPEC closest to English; these languages are typologically and genetically similar to English. A likely explanation is that models and their commonly used hyperparameters have been tuned towards English, another is that English is the most common source language in Europarl before translation.

7.4.2 n-gram versus LSTM

As expected, the LSTM outperforms the baseline n-gram models across the board. In addition, however, n-gram modeling yields relatively poor performance on some languages, such as Dutch, that have only modestly more complex inflectional morphology than English. Other phenomena—e.g., perhaps, compounding—may also be poorly modeled by n-grams.

7.4.3 The Impact of Inflectional Morphology

Another major take-away is that rich inflectional morphology seems to be a difficulty for both n-gram and LSTM LMs when measured by MCC (Section 7.1.1).

In this section we give numbers for the LSTMs. Studying Fig. 7-1a, we find that Spearman's rank correlation between a language's BPEC and its MCC is quite high ($\rho = 0.59$, significant at $p < 0.005$; a linear regression estimates a slope of 0.00062). This clear correlation between the level of inflectional morphology and the LSTM performance indicates that character-level models do not automatically fix the problem of morphological richness. If we lemmatize the words, however (Fig. 7-1b), the correlation becomes insignificant and in fact slightly negative ($\rho = -0.13$, $p \approx 0.56$; this time a linear regression estimates a slope of -0.000056), which comes as no surprise given that languages with rich morphology lose a lot of information that languages like English still retain and that thus still needs to be modeled.[7] The difference of the two previous graphs (Fig. 7-1c) shows more clearly that the LM penalty for modeling inflectional endings is greater for languages with higher MCC. See also Fig. 7-2.

The differences in BPEC among languages are reduced when we lemmatize, with standard deviation dropping from 0.065 bits to 0.039 bits. Zooming in on Finnish (see Table 7-I), we see that Finnish forms are harder to model than English forms, but Finnish lemmata are *easier* to model than English ones. We could see this as evidence that it was primarily the inflectional morphology, which lemmatization strips, that caused the differences in the model's performance on these two languages.

In conclusion, in highly inflected languages, either the utterances have more content or the models are worse. (1) Text in highly inflected languages may be *inherently harder to predict* (higher entropy per utterance) if its extra morphemes carry additional, unpredictable

[7] One might have expected *a priori* that some difference would remain, because most highly inflected languages can also vary word order to mark a topic-focus distinction, and this (occasional) marking is preserved in our experiment.

information such as gender or evidentiality. (2) Alternatively, perhaps the extra morphemes are *predictable in principle*—for example, redundant marking of grammatical number on both subjects and verbs, or marking of object case even when it is predictable from semantics or word order—and yet our current language modeling technology fails to predict them. This might happen because (2a) the technology is biased toward modeling words or characters and fails to discover intermediate morphemes, or because (2b) it fails to capture the syntactic and semantic predictors that govern the appearance of the extra morphemes. We leave it to future work to try to come up with clever setups to attempt to tease apart these hypotheses.

7.5 Related Work

Recurrent neural language models can effectively learn complex dependencies, even in open-vocabulary settings (Hwang and Sung, 2017; Kawakami et al., 2017). Whether the models are able to learn particular syntactic interactions is an intriguing question, and some methodologies have been presented to tease apart under what circumstances variously-trained models encode attested interactions (Linzen et al., 2016b; Enguehard et al., 2017). While the sort of detailed, construction-specific analyses in these papers is surely informative, our evaluation is language-wide.

MT researchers have investigated whether an English sentence contains enough information to predict the fine-grained inflections used in its foreign-language translations (see Kirov et al., 2017).

Sproat et al. (2014) present a corpus of close translations of sentences in typologically diverse languages along with detailed morphosyntactic and morphosemantic annotations, as the means for assessing linguistic complexity for comparable messages, though they expressly do not take an information-theoretic approach to measuring complexity. In the linguistics literature, McWhorter (2001) argues that certain languages are less complex than

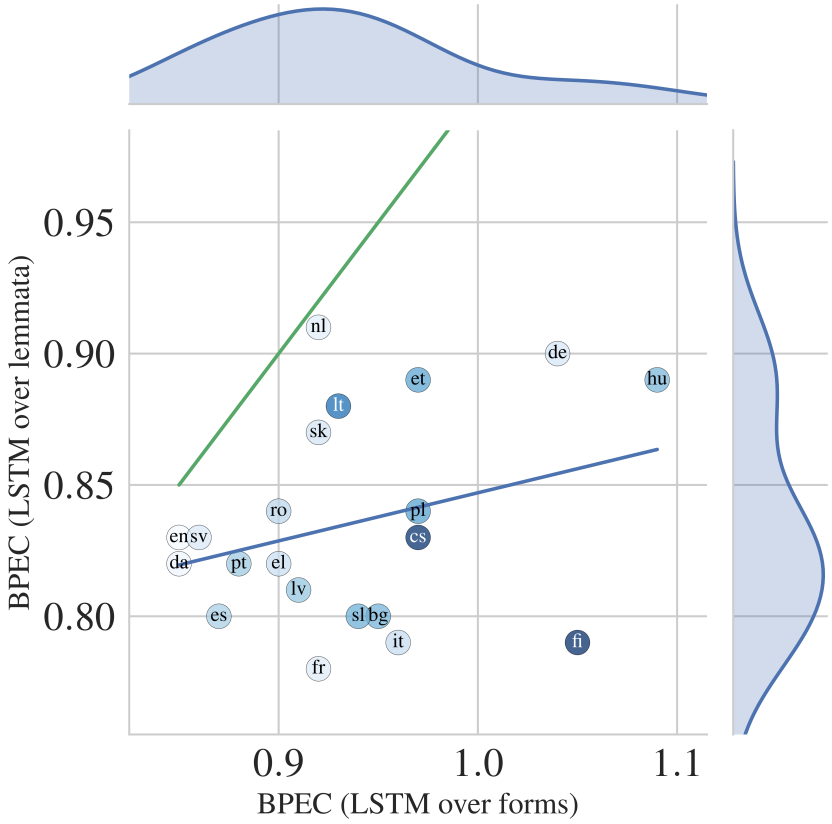


Figure 7-2. Each dot is a language, and its coordinates are the BPEC values for the LSTM LMs over words and lemmata. The top and right margins show kernel density estimates of these two sets of BPEC values. All dots follow the blue regression, but stay below the green line ($y = x$), and the darker dots—which represent languages with higher counting complexity—tend to fall toward the right but not toward the top, since counting complexity is correlated only with the BPEC over words.

others: he claims that Creoles are simpler. Müller et al. (2012) compare LMs on EuroParl, but do not compare performance across languages.

7.6 Conclusion

We have presented a simple and clean method for the cross-linguistic comparison of language modeling: we assess whether a language modeling technique can compress a sentence and its translations equally well. We show an interesting correlation between the morphological richness of a language and the performance of the model. We also run

our models on lemmatized versions of the corpora, showing that, upon the removal of inflection, no such correlation between initial morphological richness and LM performance exists. It is still unclear, however, whether the performance difference originates from the inherent difficulty of the languages or from the models—and in fact even the claim of inflectional morphology being a clear culprit will fall apart when we design a bigger study in the next chapter to cover more languages and have more sophisticated difficulty evaluation, also allowing us to revisit the question of whether Translationese is actually simpler to model.

Chapter 8

What Kind of Language Is Hard to Language-Model?

Published research and text

This chapter is largely taken from Mielke et al. (2019).

In the previous chapter we attempted to address the question of whether different languages are differently easy or hard to language model, and observed that recurrent neural network language models do not perform equally well over all the high-resource European languages found in the Europarl corpus. We speculated that inflectional morphology may be the primary culprit for the discrepancy. In this chapter, we extend these earlier experiments to cover 69 languages from 13 language families using a multilingual Bible corpus alongside Europarl, hoping to determine more accurately whether there are typological properties that make certain languages harder to language-model than others.

Methodologically, we introduce a new paired-sample multiplicative mixed-effects model to obtain language difficulty coefficients from at-least-pairwise parallel corpora, no longer strictly requiring true multi-text as in Chapter 7 for evaluation (though we still require it for fair training of models). In other words, our new model is aware of inter-sentence variation and can handle missing data. We motivate this model by considering that a language model's surprisal on a sentence as defined in Section 2.4 reflects not only the length and complexity of the specific sentence, but also the general difficulty that the model has in

predicting sentences of that language. Given language models of diverse languages, we jointly recover each language's difficulty parameter through this regression model. Our regression formula explains the variance in the dataset better than previous approaches and can also deal with missing translations for some purposes.

Given the difficulty estimates produced by this model, we conduct a correlational study, asking which typological features of a language are predictive of modeling difficulty. Our results suggest that simple properties of a language—the word inventory and (to a lesser extent) the raw character sequence length—are statistically significant indicators of modeling difficulty within our large set of languages. Interestingly, we fail to reproduce[1] the results from Chapter 7, which suggested morphological complexity as an indicator of modeling complexity. In fact, we find no tenable correlation to a wide variety of typological features, taken from the WALS dataset and other sources. Instead, we reveal far simpler statistics of the data that seem to drive complexity in a much larger sample.

On the bright side, exploiting our model's ability to handle missing data, we can directly test the hypothesis that translationese leads to easier language-modeling (Baker, 1993; Lembersky et al., 2012) and ultimately cast doubt on this claim, showing that, under the strictest controls, translationese is *different*, but not any *easier* to model according to our notion of difficulty.

We conclude with a recommendation: The world being small, questions of typology in which languages are data points are in practice small-data problems. There is a real danger that cross-linguistic studies will under-sample and thus over-extrapolate. We outline directions for future, more robust, investigations, and further caution that future

[1]We could certainly **replicate** those results in the sense that, using the surprisals from those experiments, we achieve the same correlations. However, we did not **reproduce** the results under new conditions (Drummond, 2009). Our new conditions included a larger set of languages, a more sophisticated difficulty estimation method, and—perhaps crucially—improved language modeling families that tend to achieve better surprisals (or equivalently, better perplexity). We could of course answer these questions by isolating all these factors, starting perhaps with the language models by running the analyses from Chapter 7 on the data from Chapter 7 with the *models* from this chapter, provided we can recreate all these perfectly and invest the time and compute.

work of this sort should focus on datasets with even more languages, something our new methods now allow.

8.1 The Surprisal of a Sentence

Our setup so far is much the same as in Chapter 7, using multi-text and language model evaluation metrics not normalized by length. We will introduce some more formal notation this time around to aid in the definition of our now more elaborate regression model.

8.1.1 Multi-text for a Fair Comparison

To attempt a fair comparison, we for now again make use of **multi-text**—sentence-aligned[2] translations of the *same content* in multiple languages. Different surprisals on the translations of the same sentence reflect quality differences in the language models, unless the translators added or removed information.[3]

In what follows, we will distinguish between the i^{th} **sentence** in language j, which is a specific string s_{ij}, and the i^{th} **intent**, the shared abstract thought that gave rise to all the sentences $s_{i1}, s_{i2}, \ldots$. Furthermore, instead of always assuming a fully parallel corpus as in Chapter 7, the models we build in Section 8.2 will be able to deal with missing data, which will help us to estimate the effects of translationese (Section 8.6).

8.1.2 Comparing Surprisal Across Languages

As introduced in Section 2.1, a language model p assigns a test sentence s_{ij} its surprisal $\text{NLL}(s_{ij}) = -\log_2 p(s_{ij})$. Long or unusual sentences tend to have high surprisal—but high

[2]Both corpora we use align small paragraphs instead of sentences, but for simplicity we will call them "sentences."

[3]A translator might add or remove information out of helpfulness, sloppiness, showiness, consideration for their audience's background knowledge, or deference to the conventions of the target language. For example, English conventions make it almost obligatory to express number (via morphological inflection), but make it optional to express evidentiality (e.g., via an explicit modal construction); other languages are different.

surprisal can also reflect a language's model's *failure to anticipate* predictable words.

Concretely, recall that s_{ij} and $s_{ij'}$ should contain, at least in principle, the same information for two languages j and j'—they are translations of each other. But, if we find that $\text{NLL}(s_{ij}) > \text{NLL}(s_{ij'})$, we must assume that either s_{ij} contains more information than $s_{ij'}$, or that our language model was simply able to predict it less well.[4] If we were to assume that our language models were perfect in the sense that they captured the true probability distribution of a language, we could make the former claim; but we suspect that much of the difference can be explained by our imperfect LMs rather than inherent differences in the expressed information (see the discussion in Footnote 3).

8.1.3 Our Language Models

Specifically, the crude tools we use are recurrent neural network language models (RNNLMs) over characters and BPE subword units. Again, for fairness, it is of utmost importance that these language models are **open-vocabulary**, i.e., they predict the entire string and cannot cheat by predicting only UNK ("unknown") for some words of the language.[5]

The removal of the n-gram model from our experimental setup in place of BPE-based models follows popular usage of these models in recent years, leaving us with these two model choices as defined in Section 4.2:

character-level LSTM LM (henceforth *Char-RNNLM*) An obvious drawback of the model is that it has no explicit representation of reusable substrings (see arguments in Part II), but the fact that it does not rely on a somewhat arbitrary word segmentation or tokenization makes it attractive for this study.

[4]The former might be the result of overt marking of, say, evidentiality or gender, which adds information. We hope that these differences are taken care of by diligent translators producing faithful translations in our multi-text corpus.

[5]We restrict the set of characters to those that we see at least 25 times in the training set, replacing all others with a new symbol ★ as discussed in Section 4.2. We make an exception for Chinese, where we only require each character to appear at least twice. These thresholds result in negligible "out-of-alphabet" rates for all languages.

BPE-token-level LSTM LM (henceforth *BPE-RNNLM*) Recall that by specifying the number of BPE merges, we can effectively interpolate between a word-level model (∞ merges) and a kind of character-level model (0 merges). As Fig. 8-1 shows, the number of merges that maximizes log-likelihood of our dev set differs from language to language.[6] However, as we will see in Fig. 8-11, tuning this parameter does not substantially influence our results. We therefore will refer to the model with $0.4|\mathcal{V}|$ merges as BPE-RNNLM. It is worth pointing out that that choice, while convenient in this visualization, may lead to an unreliable conclusion that we will discuss in Section 8.5.6.

8.2 Aggregating Sentence Surprisals

In Chapter 7 we evaluated the model for language j simply by its total surprisal $\sum_i \text{NLL}(s_{ij})$. This comparative measure required a complete multi-text corpus containing every sentence s_{ij} (the expression of the intent i in language j) for evaluation. We relax this requirement by using a fully probabilistic regression model that can deal with missing data (Fig. 8-2) for the evaluation itself.[7] Our model predicts each sentence's surprisal $y_{ij} = \text{NLL}(s_{ij})$ using an intent-specific "information content" factor n_i, which captures the inherent surprisal of the intent, combined with a language-specific difficulty factor d_j. This represents a better approach to varying sentence lengths and lets us work with missing translations in the test data (though it does not remedy our need for fully parallel language model training data).

We will now define a series of models, increasing in complexity and beginning with a basic multiplicative mixed-effects model that is then extended to correctly handle the variance of longer and shorter sentences as well as outliers.

[6] Fig. 8-1 shows the 21 languages of the Europarl dataset. Optimal values: 0.2 (et); 0.3 (fi, lt); 0.4 (de, es, hu, lv, sk, sl); 0.5 (da, fr, pl, sv); 0.6 (bg, ru); 0.7 (el); 0.8 (en); 0.9 (it, pt).

[7] Specifically, we deal with data missing completely at random (MCAR), a strong assumption on the data generation process. More discussion on this can be found in Section 8.2.6.

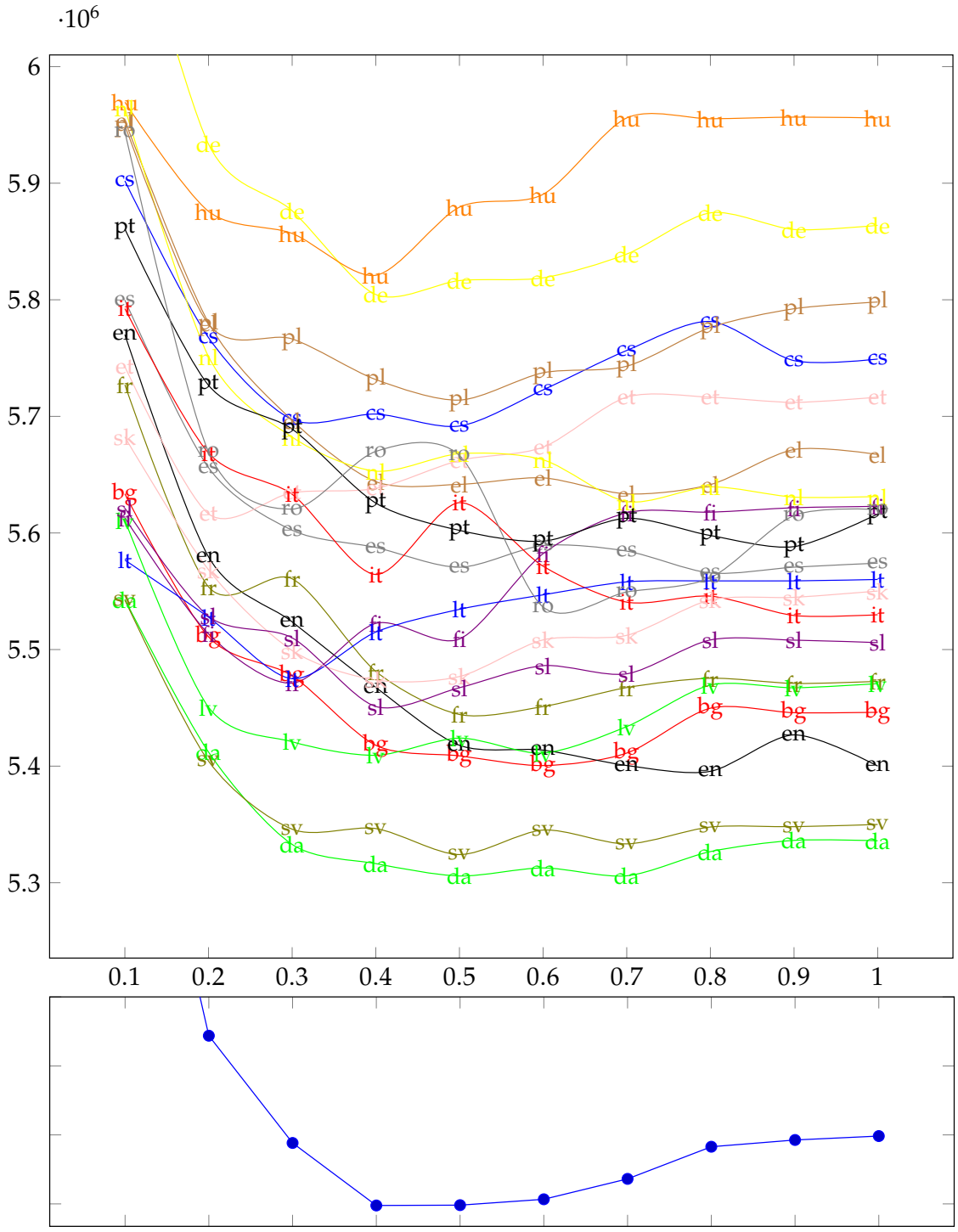


Figure 8-1. Top: For each language, total NLL of the dev corpus varies with the number of BPE merges, which is expressed on the x-axis as a fraction of the number of observed word types $|\mathcal{V}|$ in that language.[6] Bottom: Averaging over all 21 languages motivates a global value of 0.4.

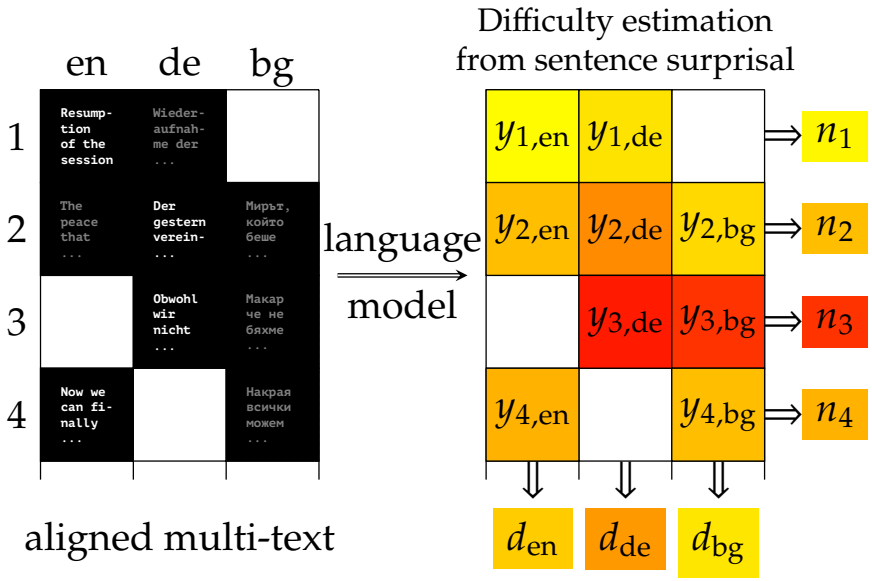


Figure 8-2. Jointly estimating the information n_i present in each multi-text intent i *and* the difficulty d_j of each language j. At left, gray text indicates translations of the original (white) sentence in the same row. At right, darker cells indicate higher surprisal/difficulty. Empty cells indicate missing translations. English (en) is missing a hard sentence and Bulgarian (bg) is missing an easy sentence, but this does not mislead our method into estimating English as easier than Bulgarian.

8.2.1 Model 1: Multiplicative Mixed-effects

Model 1 is a multiplicative mixed-effects model:

$$y_{ij} = n_i \cdot \exp(d_j) \cdot \exp(\epsilon_{ij}) \tag{8.1}$$

$$\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2) \tag{8.2}$$

This says that each intent i has a latent **size** of n_i—measured in some abstract "informational units"—that is observed indirectly in the various sentences s_{ij} that express the intent. Larger n_i tend to yield longer sentences. Sentence s_{ij} has y_{ij} bits of surprisal; thus the multiplier y_{ij}/n_i represents the number of bits that *language* j used to express each informational unit of intent i, under our language model of language j. Our mixed-effects model assumes that this multiplier is log-normally distributed over the sentences i: that is, $\log(y_{ij}/n_i) \sim \mathcal{N}(d_j, \sigma^2)$, where mean d_j is the **difficulty** of language j. That is,

$y_{ij}/n_i = \exp(d_j + \epsilon_{ij})$ where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$ is residual noise, yielding Eqs. (8.1) to (8.2).[8] We jointly fit the intent sizes n_i and the language difficulties d_j, as well as σ.

8.2.2 Model 2: Heteroscedasticity

Because it is multiplicative, Model 1 appropriately predicts that in each language j, intents with large n_i will not only have larger y_{ij} values but these values will vary more widely. However, Model 1 is **homoscedastic**: the variance σ^2 of $\log(y_{ij}/n_i)$ is assumed to be independent of the independent variable n_i, which predicts that the distribution of y_{ij} should spread out *linearly* as the information content n_i increases: e.g., $p(y_{ij} \geq 13 \mid n_i = 10) = p(y_{ij} \geq 26 \mid n_i = 20)$. That assumption is questionable, since for a longer sentence, we would expect $\log y_{ij}/n_i$ to come closer to its mean d_j as the random effects of individual translational choices average out.[9]

Figure 8-3 shows visualizations of this issue: from a quick bird's-eye view we can see that the variance of this multiplier on the y-axis does indeed shrink with higher n_i on the x-axis and that the effect mostly disappears for the average (final row at the bottom).

We address this issue by assuming that y_{ij} results from $n_i \in \mathbb{N}$ independent choices:

$$y_{ij} = \exp(d_j) \cdot \left(\sum_{k=1}^{n_i} \exp \epsilon_{ijk} \right) \tag{8.3}$$

$$\epsilon_{ijk} \sim \mathcal{N}(0, \sigma^2) \tag{8.4}$$

The number of bits for the k^{th} informational unit now varies by a factor of $\exp \epsilon_{ijk}$ that is log-normal and independent of the other units. It is common to approximate the sum of independent log-normals by another log-normal distribution, matching mean and variance

[8] It is tempting to give each language its own σ_j^2 parameter, but then the MAP estimate is pathological, since infinite likelihood can be attained by setting one language's σ_j^2 to 0.

[9] Similarly, flipping a fair coin 10 times results in 5 ± 1.58 heads where 1.58 represents the standard deviation, but flipping it 20 times does not result in $10 \pm 1.58 \cdot 2$ heads but rather $10 \pm 1.58 \cdot \sqrt{2}$ heads. Thus, with more flips, the ratio heads/flips tends to fall closer to its mean 0.5.

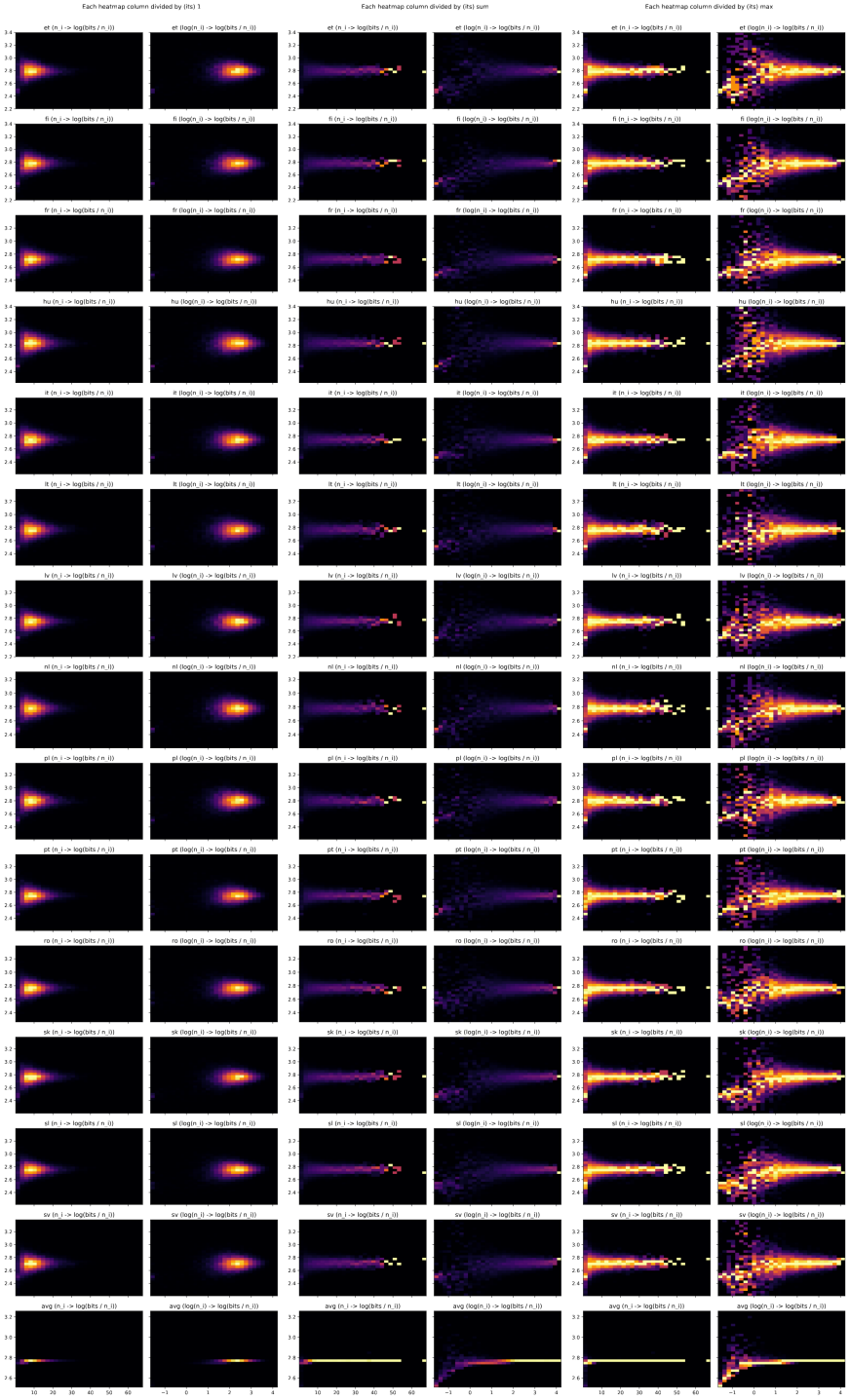


Figure 8-3. Variance in y_{ij} on the y-axis for varying n_i on the x-axis. From left to right: global heatmap, normalizing in each column, and normalizing by the max, not the sum; pairing n_i and $\log n_i$ left and right in each column. Only a subset of Europarl languages and their average is shown for space reasons.

(Fenton-Wilkinson approximation; Fenton, 1960),[10] yielding Model 2:

$$y_{ij} = n_i \cdot \exp(d_j) \cdot \exp(\epsilon_{ij}) \tag{8.5}$$

$$\sigma_i^2 = \ln\left(1 + \frac{\exp(\sigma^2)-1}{n_i}\right) \tag{8.6}$$

$$\epsilon_{ij} \sim \mathcal{N}\left(\frac{\sigma^2-\sigma_i^2}{2},\ \sigma_i^2\right), \tag{8.7}$$

in which the noise term ϵ_{ij} now depends on n_i. Unlike (8.4), this formula no longer requires $n_i \in \mathbb{N}$; we allow any $n_i \in \mathbb{R}_{>0}$, which will also let us use gradient descent in estimating n_i.

In effect, fitting the model chooses each n_i so that the resulting intent-specific but language-independent distribution of $n_i \cdot \exp(\epsilon_{ij})$ values,[11] after it is scaled by $\exp(d_j)$ for each language j, will assign high probability to the observed y_{ij}. Notice that in Model 2, the scale of n_i becomes meaningful: fitting the model will choose the size of the abstract informational units so as to predict how rapidly σ_i falls off with n_i. This contrasts with Model 1, where doubling all the n_i values could be compensated for by halving all the $\exp(d_j)$ values.

8.2.3 Model 2L: An Outlier-Resistant Variant

Consider now the problem of outliers. In some cases, sloppy translation will yield a y_{ij} that is unusually high or low given the y'_{ij} values of other languages j'. Such a y_{ij} is not good evidence of the quality of the language model for language j since it has been corrupted by the sloppy translation. However, under Model 1 or 2, we could not simply explain this corrupted y_{ij} with the random residual ϵ_{ij} since large $|\epsilon_{ij}|$ is highly unlikely under the Gaussian assumption of those models. Rather, y_{ij} would have significant influence on our estimate of the per-language effect d_j.

This is the usual motivation for switching to L1 regression, which replaces the Gaussian

[10] There are better approximations, but even the only slightly more complicated Schwartz-Yeh approximation (Schwartz and Yeh, 1982) already requires costly and complicated approximations in addition to lacking the generalizability to non-integral n_i values that we will obtain for the Fenton-Wilkinson approximation.

[11] The distribution of ϵ_{ij} is the same for every j. It no longer has mean 0, but it depends only on n_i.

prior on the residuals with a Laplace[12] prior.[13] Using this Laplace distribution instead of a Gaussian in (8.7) as an approximation to the distribution of ϵ_{ij}. The Laplace distribution is heavy-tailed, so it is more tolerant of large residuals. We choose its two parameters exactly as in (8.7). This heavy-tailed ϵ_{ij} distribution can be viewed as approximating a version of Model 2 in which the ϵ_{ijk} themselves follow some heavy-tailed distribution.

8.2.4 Regression, Model 3: Handling outliers cleverly

To tackle the problem of outliers, we also posit yet another model that tries to hone in on the reasons for outliers. First let us identify two failure modes:

(a) part of a sentence was omitted (or added) during translation, changing the n_i additively; thus we should use a noisy $n_i + v_{ij}$ in place of n_i in Eqs. (8.1) and (8.6)

(b) the style of the translation was unusual throughout the sentence; thus we should use a noisy $n_i \cdot \exp v_{ij}$ instead of n_i in Eqs. (8.1) and (8.6)

In both cases $v_{ij} \sim \text{Laplace}(0, b)$, i.e., v_{ij} specifies sparse (in the case of MAP, which we will eventually choose to perform) additive or multiplicative noise in v_{ij} (on language j only).[14]

Let us write out version (b), which is a modification of Model 2 (Eqs. (8.1), (8.6) and (8.7)):

$$\begin{aligned} y_{ij} &= (n_i \cdot \exp v_{ij}) \cdot \exp(d_j) \cdot \exp(\epsilon_{ij}) \\ &= n_i \cdot \exp(d_j) \cdot \exp(\epsilon_{ij} + v_{ij}) && (8.8) \\ v_{ij} &\sim \text{Laplace}(0, b) && (8.9) \\ \sigma_i^2 &= \ln\left(1 + \tfrac{\exp(\sigma^2)-1}{n_i \cdot \exp v_{ij}}\right) && (8.10) \\ \epsilon_{ij} &\sim \mathcal{N}\left(\tfrac{\sigma^2-\sigma_i^2}{2},\ \sigma_i^2\right), && (8.11) \end{aligned}$$

[12] One could also use a Cauchy distribution instead of the Laplace distribution to get even heavier tails, but we saw little difference between the two in practice.

[13] An alternative would be to use a method like RANSAC to discard y_{ij} values that do not appear to fit.

[14] However, version (a) is then deficient since it then incorrectly allocates some probability mass to $n_i + v_{ij} < 0$ and thus $y_{ij} < 0$ is possible. This could be fixed by using a different sparsity-inducing distribution.

Comparing Eq. (8.8) to Eq. (8.1), we see that we are now modeling the residual error in $\log y_{ij}$ as a *sum of two noise terms* $a_{ij} = \nu_{ij} + \epsilon_{ij}$ and penalizing it by (some multiple of) the weighted sum of $|\nu_{ij}|$ and ϵ_{ij}^2, where large errors can be more cheaply explained using the former summand, and small errors using the latter summand.[15] The weighting of the two terms is a tunable hyperparameter.

We did implement this model and test it on data, but not only was fitting it much harder and slower, it also did not yield particularly encouraging results, leading us to omit it from all subsequent discussions.

8.2.5 Estimating model parameters

8.2.5.1 MAP inference

We fit each regression model's parameters by L-BFGS. We then evaluate the model's fitness by measuring its held-out data likelihood—that is, the probability it assigns to the y_{ij} values for held-out intents i. Here we use the previously fitted d_j and σ parameters, but we must newly fit n_i values for the new i using MAP estimates or posterior means.

8.2.5.2 Bayesian inference through HMC

As our model of y_{ij} values is fully generative, one could place priors on our parameters and do full inference of the posterior rather than performing MAP inference. We did experiment with priors but found them so quickly overwhelmed by the data that it did not make much sense to spend time on them, opting for uninformative priors in our final implementation. This implementation was done in STAN (Carpenter et al., 2017), a toolkit for fast, state-of-the-art inference using Hamiltonian Monte Carlo (HMC) estimation.

[15] The cheapest penalty or explanation of the weighted sum $\delta|\nu_{ij}| + \frac{1}{2}\epsilon_{ij}^2$ for some weighting or threshold δ (which adjusts the relative variances of the two priors) is $\nu = 0$ if $|a| \leq \delta$, $\nu = a - \delta$ if $a \geq \delta$, and $\nu = -(a - \delta)$ if $a < -\delta$ (found by minimizing $\delta|\nu| + \frac{1}{2}(a - \nu)^2$, a convex function of ν). This implies that we incur a quadratic penalty $\frac{1}{2}a^2$ if $|a| \leq \delta$, and a linear penalty $\delta(|a| - \frac{1}{2}\delta)$ for the other cases; this penalty function is exactly the Huber loss of a, and essentially imposes an L2 penalty on small residuals and an L1 penalty on large residuals (outliers), so our estimate of d_j will be something between a mean and a median.

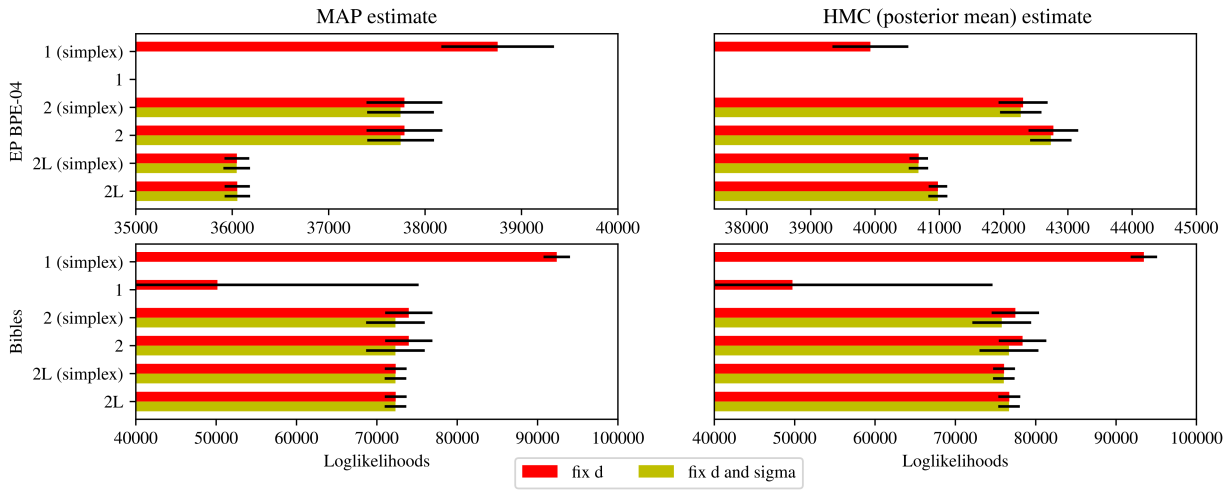


Figure 8-4. Achieved log-likelihoods of the regression model, trying to predict held-out language model surprisal values. Top: Europarl (BPE), Bottom: Bibles, Left: MAP inference, Right: HMC inference (posterior mean).

Running HMC unfortunately scales superlinearly with the number of sentences (and thus results in very long sampling times), and the posteriors we obtained were unimodal with relatively small variances (see below). We therefore work with the MAP estimates in Model 2 in the rest of this chapter.

All scripts used in this chapter including the STAN implementation generator are available at https://github.com/sjmielke/whatkind-scripts.

8.2.5.3 Model Comparison / Goodness of Fit

A full comparison of our models under various conditions can be found in Fig. 8-4, which shows the log-probability of held-out data (i.e., unseen sentences' language model surprisals) under the regression model, by fixing the estimated difficulties d_j (and sometimes also the estimated variance σ^2) to their values obtained from the matrix of parallel sentences, and then finding either MAP estimates or posterior means (by running HMC using STAN) of the other parameters, in particular n_i for the new sentences i. The error bars are the standard deviations when running the model over different subsets of surprisal data.

The "simplex" version of a model in Fig. 8-4 is defined to force all $\exp d_j$ to add up to

the number of languages (i.e., encouraging each one to stay close to 1). This is *necessary* for Model 1, which otherwise is unidentifiable (hence the enormous standard deviation). For other models, it turns out to only have much of an effect on the posterior means, not on the log-probability of held out data under the MAP estimate. For stability, we in all cases take the best result when initializing the new parameters randomly or "sensibly," i.e., the n_i of an intent i is initialized as the average of the corresponding sentences' y_{ij}.

The primary findings are as follows. On Europarl data (which has fewer languages), Model 2 performs best. On the Bible corpora, all models are relatively close to one another, though the robust Model 2L gets more consistent results than Model 2 across data subsets. We use MAP estimates under Model 2 for all remaining experiments for speed and simplicity.

8.2.6 A Note on Missing Data

We stated in the beginning of this section that our model can deal with missing data, but this is true only for the case of data **missing completely at random** (MCAR), the strongest assumption we can make about missing data: the missingness of data is neither influenced by what the value would have been (had it not been missing), nor by any covariates.

Sadly, this assumption is rarely met in real translations, where difficult, useless, or otherwise *distinctive* intents may be skipped. This leads to data **missing at random** (MAR), where the missingness of a translation is correlated with the original sentence it should have been translated from. This is a case that is easy enough to imagine: perhaps some section are too hard to translate, were not necessary to translate or are missing for other reasons (even as mundane as: an off-by-one error always drops the last section, i.e., intent, of a session)

Less likely, but still technically possible and worth mentioning is data **missing not at random** (MNAR), where the missingness of a translation is correlated with *that translation itself*, i.e., the original sentence was translated, but the translation was then deleted for

a reason that depends on that translation rather than the original sentence or shared intent. Coming up with a way this could occur requires more handwaving, perhaps about censorship or about even more obscure bugs like perhaps characters occuring in some languages and some sentences somehow corrupting the database, leading to the sentence deletion.

Either way, to avoid all these concerns, we use fully parallel data where possible; in fact, we only make use of the ability to deal with missing data in Section 8.6.[16]

8.3 Languages and Data Selection

Having outlined our method for estimating language difficulty scores d_j, we now seek data to do so for all our languages. If we wanted to cover the most languages possible with parallel text, we should surely look at the Universal Declaration of Human Rights, which has been translated into over 500 languages. Yet this short document is far too small to train state-of-the-art language models. In this chapter, we will therefore follow previous work in using the Europarl corpus (Koehn, 2005), but also for the first time make use of 106 Bibles from Mayer and Cysouw (2014)'s corpus.

Although our regression models of the surprisals y_{ij} can be estimated from incomplete multi-text, the surprisals themselves are derived from the language models we are comparing. To ensure that the language models are comparable, we want to train them on completely parallel data in the various languages. For this, we seek complete multi-text.

8.3.1 Europarl: 21 Languages

The Europarl corpus (Koehn, 2005) contains decades worth of discussions of the European Parliament, where each intent appears in up to 21 languages; we already used it in Chapter 7 for its size and stability. In Section 8.6, we will also exploit the fact that each intent's original

[16]Note that this application counts as data MAR and not MCAR, thus technically violating our requirements, but only in a minor enough way that we are confident it can still be applied.

language is known. To simplify our access to this information, we will use the "Corrected & Structured Europarl Corpus" (CoStEP) corpus (Graën et al., 2014), a structured version of Europarl, encoded in XML. From it, we extract the intents that appear in all 21 languages, as enumerated in Footnote 6 on page 143.

8.3.1.1 Extraction Process

In the "Corrected & Structured Europarl Corpus" (CoStEP) corpus (Graën et al., 2014), sessions are grouped into *turns*, each turn has one speaker (that is marked with clean attributes like native language) and a number of aligned *paragraphs* for each language, i.e., the actual multi-text. We call each such paragraph a *sentence* for simplicity and see all sentences that are translations of each other as a realization of a single *intent* – splitting into actual sentences is a noisy and thus risky process that we can forego by simply training on more than one sentence at a time.

The big issue is that the paragraphs are not properly aligned between languages; that is, a turn may contain a different number of paragraphs in each language (with some content missing in some languages), and worse, some turns are *wrongly* aligned. Graën et al. (2014) did try to correct for this, aligning turns based on speaker names and paragraph count (but not based on the content), but as we can see the process did leave some errors, leaving us with the choice of:

1. adding more sophisticated fixing, e.g., re-aligning turns by *content*, emphasizing precision over recall,

2. ignoring all ill-fitting turns (i.e., unequal numbers of paragraphs within one turn), losing roughly 27% of intents,

3. ignoring all *chapters* with ill-fitting turns (because mismatches may mean misaligned turns around the faulty turn itself), losing 87%, or

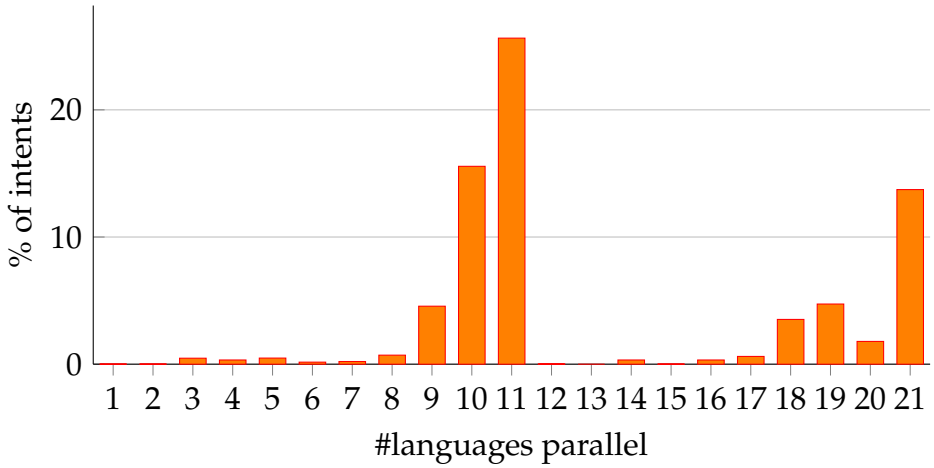


Figure 8-5. In how many languages are the intents in Europarl translated? (intents from ill-fitting turns included in 100%, but not plotted)

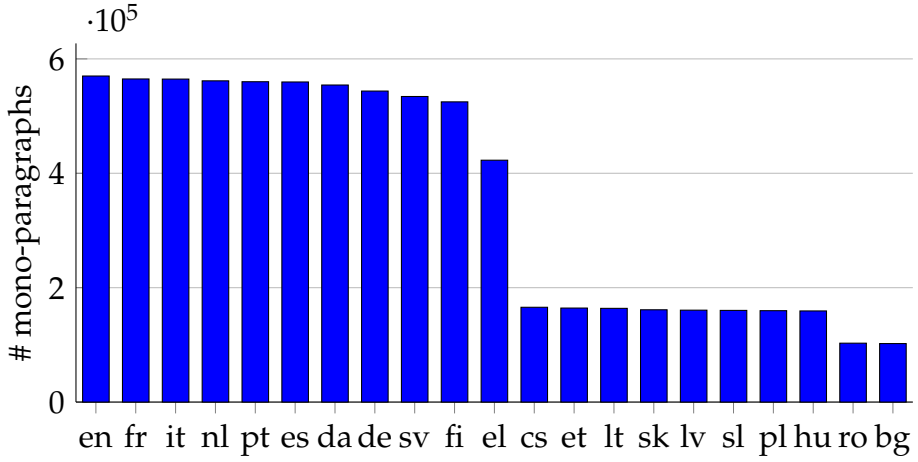


Figure 8-6. How many sentences are there per Europarl language?

4. not caring at all and instead just modeling on a turn-level instead of sentence- or paragraph-level.

We opt for the second option, ignoring all paragraphs that are in *ill-fitting* turns (i.e., turns with an unequal number of paragraphs across languages, a clear sign of an incorrect alignment), losing roughly 27% of intents. After this cleaning step, only 14% of *intents* are represented in all 21 languages, see the distribution in Fig. 8-5 (the peak at 11 languages is explained by looking at the raw number of sentences present in each language, shown in Fig. 8-6).

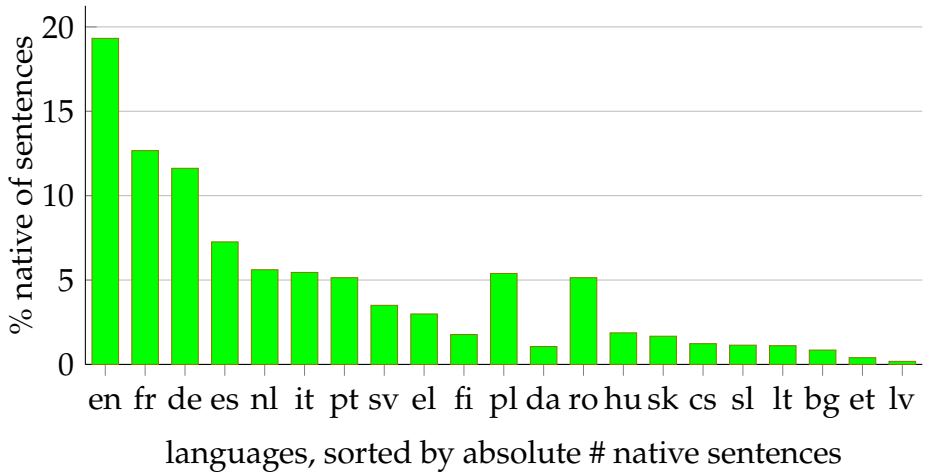


Figure 8-7. How many of the Europarl sentences in one language are "native"?

Since we want a fair comparison, we use the aforementioned 14% of Europarl, giving us 78169 intents that are represented in all 21 languages - and while we could use the partial remainder for our missing-data-aware regression model, doing so would be of little use: even when splitting the 14% into training, development, and test sets, the test set on which we perform the regression already contains over 10000 sentences – more than enough to get fairly precise estimates.

Finally, it should be said that the text in CoStEP itself contains some markup, marking reports, ellipses, etc., but we strip this additional markup to obtain the raw text. We tokenize it using the tokenizer from Section 4.1.2 and split the obtained 78169 paragraphs into training set, development set for tuning our language models, and test set for our regression, again by dividing the data into blocks of 30 paragraphs and then taking 5 sentences for the development and test set each, leaving the remainder for the training set. This way we ensure uniform division over sessions of the parliament and sizes of $^2/_3$, $^1/_6$, and $^1/_6$, respectively.

8.3.1.2 How are the source languages distributed?

An obvious question we should ask is: how many "native" sentences can we actually find in Europarl? One could assume that there are as many native *sentences* as there are *intents*

in total, but there are three issues with this: the first is that the *president* in any Europarl session is never annotated with name or native language (leaving us guessing what the native version of any president-uttered intent is; 12% of all intents in Europarl that can be extracted have this problem), the second is that a number of speakers are labeled with "unknown" as native language (10% of sentences), and finally some speakers have their native language annotated, but it is nowhere to be found in the corresponding sentences (7% of sentences).

Looking only at the native sentences that we could identify, we can see that there are native sentences in every language, but unsurprisingly, some languages are overrepresented. Dividing the number of *native* sentences in a language by the number of *total* sentences, we get an idea of how "natively spoken" the language is in Europarl, shown in Fig. 8-7.

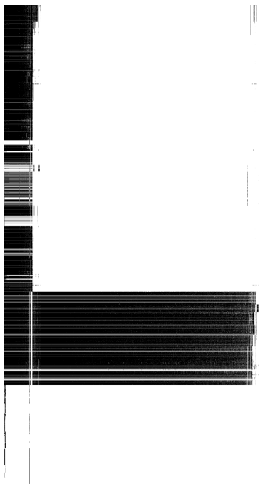
8.3.2 The Bible: 62 Languages

The Bible is a religious text that has been used for decades as a dataset for massively multilingual NLP (Resnik et al., 1999; Yarowsky et al., 2001; Agić et al., 2016). It is composed of the *Old Testament* and the *New Testament* (the latter of which has been much more widely translated), both consisting of individual *books*, which, in turn, can be separated into *chapters*, but we will only work with the smallest subdivision unit: the *verse*, corresponding roughly to a sentence (see Footnote 2).

Turning to the (already tokenized[17] and aligned) collection assembled by Mayer and Cysouw (2014), we see that it has over 1000 New Testaments, but far fewer "complete" Bibles.

Unfortunately even these "complete" Bibles are missing verses and sometimes entire books—some of these discrepancies may be reduced to the question of the legitimacy

[17]The fact that the resource is tokenized is (yet) another possible confound for this study: we are not comparing performance on languages, but on languages/Bibles *with some specific translator and tokenization*. It is possible that our y_{ij} values for each language j depend to a small degree on the tokenizer that was chosen for that language.



(a) All 1174 Bibles, in packets of 20 verses, Bibles sorted by number of verses present, verses in chronological order. The New Testament (third quarter of verses) is present in almost every Bible.

(b) The 131 Bibles with at least 20000 verses, in packets of 150 verses (this time, both sorted). The optimization task is to remove rows and columns in this picture until only black remains.

Figure 8-8. Presence (black) of verses (y-axis) in Bibles (x-axis). Both pictures are downsampled, resulting in grayscale values for all packets of N values.

of certain biblical books, others are simply artifacts of verse numbering and labeling of individual translations.

For us, this means that we can neither simply take all translations that have "the entire Bible" (in fact, no single Bible in the set covers the union of all others' verses), nor can we take all Bibles and work with the verses that they all share (because, again, no single verse is shared over all given Bibles). The whole situation is visualized in Fig. 8-8.

We have to find a tradeoff: take as many Bibles as possible that share as many verses as possible. Specifically, we cast this selection process as an optimization problem: select Bibles such that the number of verses overall (i.e., the number of verses shared times the number of Bibles) is maximal, breaking ties in favor of including more Bibles and ensuring that we have at least 20000 verses overall to ensure applicability of neural language models. This problem can be cast as an integer linear program (via a simpler QP that we may as well directly use) and solved using a standard optimization tool (Gurobi) within a few hours (Fig. 8-9).

```
# Lists of binary variables indicating presence in the solution
# (i.e., 0 or 1)
include_bible = ["bible_1", "bible_2", ...]
include_language = ["lang_eng", "lang_deu", ...]
include_verse = ["verse_1", "verse_2", ...]

# Quadratic objective including linear tie-breaking
MAXIMIZE: sum(include_language) * sum(include_verse)
            + 0.000001 * sum(include_bible)

# Aggregate bibles in the same language
# (only count English once even though it has many bibles)
for var, lang in zip(include_language, languages):
  # list of bible presence variables for this language
  bibles_in_this_language = [
    b for b, l in zip(include_bible, bible2language) if l == lang
  ]
  # var can only be set to 1 if we have at least 1 bible in this
  language
  CONSTRAINT: var <= sum(bibles_in_this_language)

# Require full parallelity
for b in bibles:
  for v in verses:
    if not presence[bible, verse]:
      # if this verse doesn't exist in this bible,
      # only one presence indicator can be part of the solution
      CONSTRAINT: include_bible[b] + include_verse[v] <= 1
```

Figure 8-9. The QP version of the optimization problem (since it is easier to read than the equivalent ILP and no harder to solve) in python-esque psuedocode.

English corpus	lines	words	chars
WikiText-103	1809468	101880752	543005627
Wikipedia ($\underset{\in[\texttt{a-z }]^*}{\texttt{text8}}$)	1	17005207	100000000
Europarl	78169	6411731	37388604
WikiText-2	44836	2507005	13378183
PTB	49199	1036580	5951345
62/106-parallel Bible	25996	~700000	~3600000

Table 8-I. Sizes of various language modeling datasets, numbers estimated using `wc`.

The optimal solution that we find contains 25996 verses for 106 Bibles in 62 languages,[18] spanning 13 language families.[19] It is unfortunate not to have more families or more languages per family, if we had more data to work with, using this as a constraint or a reward in the ILP could be a viable path forward, as well. A broader sample could of course be obtained by taking only the New Testament, but unfortunately that has < 8000 verses, a meager third of our dataset that is already smaller that the usually considered tiny PTB dataset—these sizes are illustrated for the Bibles alone in Fig. 8-10 and in relation to other datasets like the PTB dataset in Table 8-I.

We split them into train/dev/test by dividing the data into blocks of 30 paragraphs and then taking 5 sentences for the development and test set each, leaving the remainder for the training set. This way we ensure uniform division over books of the Bible and sizes of $2/3$, $1/6$, and $1/6$, respectively.

We allow j to range over the 106 Bibles, so when a language has multiple Bibles, we estimate a separate difficulty d_j for each one instead of sharing a single d. This means that each d_j is estimated from the similar amounts of data, and more importantly it will allow us to later analyze how much difficulties *within* a language but *across* translations of the Bible differ.

[18] afr, aln, arb, arz, ayr, bba, ben, bqc, bul, cac, cak, ceb, ces, cmn, cnh, cym, dan, deu, ell, eng, epo, fin, fra, guj, gur, hat, hrv, hun, ind, ita, kek, kjb, lat, lit, mah, mam, mri, mya, nld, nor, plt, poh, por, qub, quh, quy, quz, ron, rus, som, tbz, tcw, tgl, tlh, tpi, tpm, ukr, vie, wal, wbm, xho, zom

[19] 22 Indo-European, 6 Niger-Congo, 6 Mayan, 6 Austronesian, 4 Sino-Tibetan, 4 Quechuan, 4 Afro-Asiatic, 2 Uralic, 2 Creoles, 2 Constructed languages, 2 Austro-Asiatic, 1 Totonacan, 1 Aymaran; we are reporting the first category on Ethnologue (Paul et al., 2009) for all languages, manually fixing tlh ↦ Constructed language.

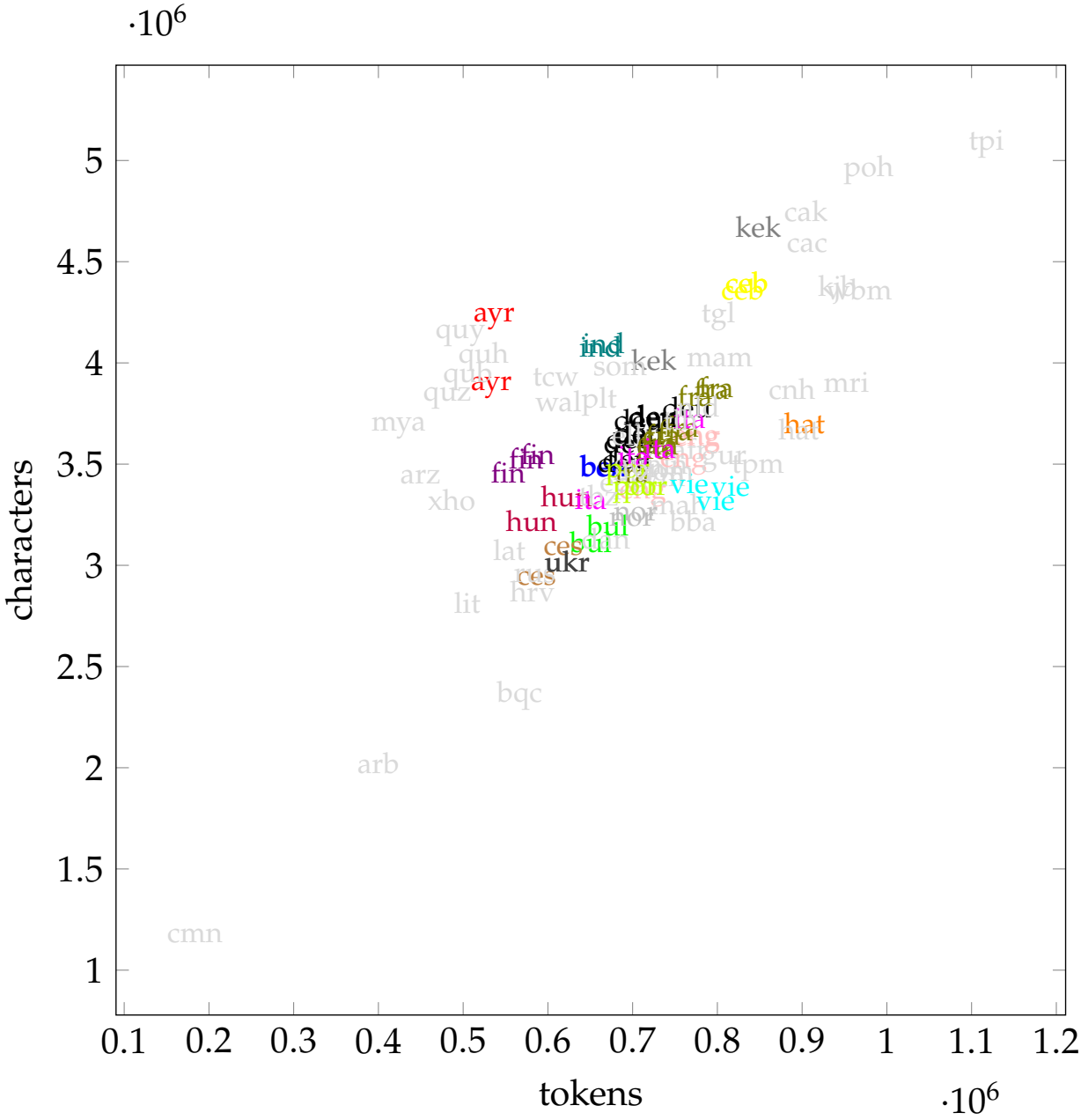


Figure 8-10. Tokens and characters (as reported by `wc -w/-m`) of the 106 Bibles. Equal languages share a color, all others are shown in faint gray. Most Bibles have around 700k tokens and 3.6M characters; outliers like Mandarin Chinese (cmn) are not surprising.

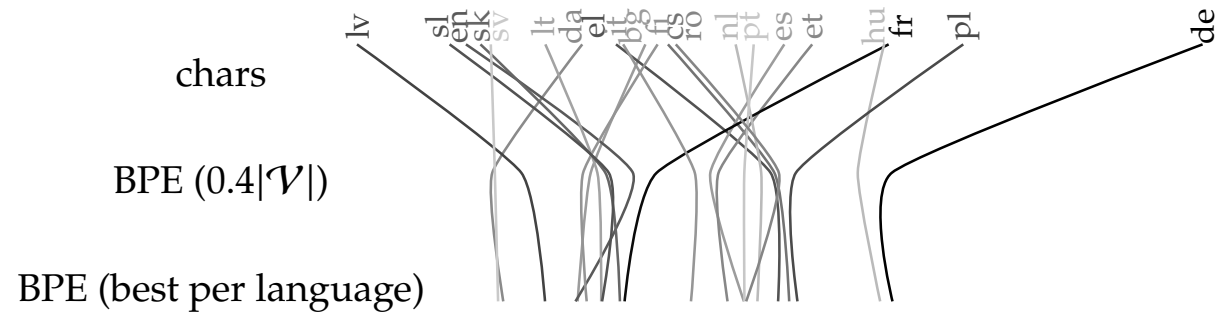


Figure 8-11. The Europarl language difficulties (ascending left-to-right) appear more similar, and are ordered differently, when the RNN models use BPE units instead of character units. Tuning BPE per-language has a small additional effect.

8.4 The Difficulties of 69 languages

We use $2/3$ of the raw data to train our models, $1/6$ to tune them and the remaining $1/6$ to test them.

8.4.1 Overall Results

The estimated difficulties are visualized in Fig. 8-12. We can see that general trends are preserved between datasets: German and Hungarian are hardest, English and Lithuanian easiest. As we can see in Fig. 8-11 for Europarl, the difficulty estimates are hardly affected when tuning the number of BPE merges per-language instead of globally, validating our approach of using the BPE model for our experiments. A bigger difference seems to be the choice of char-RNNLM vs. BPE-RNNLM, which changes the ranking of languages both on Europarl data and on Bibles. We still see German as the hardest language, but almost all other languages switch places. Specifically, we can see that the variance of the char-RNNLM is much higher.

8.4.2 Are All Translations Equal?

Texts like the Bible are justly infamous for their sometimes archaic or unrepresentative use of language. The fact that we sometimes have multiple Bible translations in the same

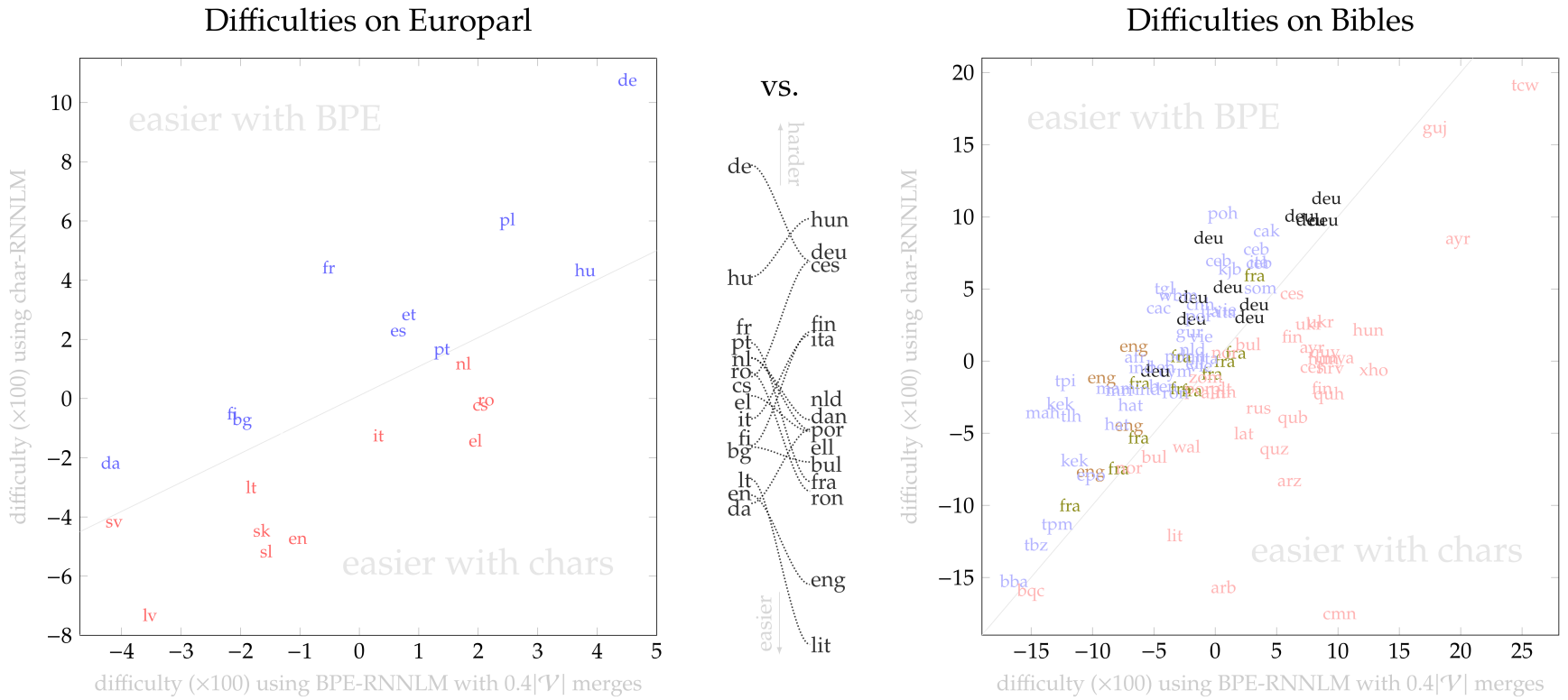


Figure 8-12. Difficulties of 21 Europarl languages (left) and 106 Bibles (right), comparing difficulties when estimated from BPE-RNNLMs vs. char-RNNLMs. Highlighted on the right are deu and fra, for which we have many Bibles, and eng, which has often been prioritized even over these two in research. In the middle we see the difficulties of the 14 languages that are shared between the Bibles and Europarl aligned to each other (averaging all estimates), indicating that the general trends we see are not tied to either corpus.

language lets us observe variation by translation style.

The sample-based standard deviation estimate of d_j among the 106 Bibles j is 0.076/0.063 for BPE/char-RNNLM. Within the 11 German, 11 French, and 4 English Bibles, the sample-based standard deviations estimate were roughly 0.05/0.04, 0.05/0.04, and 0.02/0.04 respectively: so style accounts for less than half the variance. We also consider another parallel corpus, created from the NIST OpenMT competitions on machine translation, in which each sentence has 4 English translations (NIST Multimodal Information Group, 2010a,b,c,d,e,f,g, 2013b,a). We get a sample-based standard deviation estimate of 0.01/0.03 among the 4 resulting English corpora, suggesting that language difficulty estimates (particularly the BPE estimate) depend less on the translator, to the extent that these corpora represent individual translators.

8.5 What Correlates with Difficulty?

Making use of our results on these languages, we can now answer the question: what features of a language correlate with the difference in language complexity?[20] Sadly, we cannot conduct all analyses on all data: the Europarl languages are well-served by existing tools like UDPipe (Straka et al., 2016), but the languages of our Bibles are often not. We therefore conduct analyses that rely on automatically extracted features only on the Europarl corpora. Note that to ensure a false discovery rate of at most $\alpha = .05$, all reported p-values have to be corrected using Benjamini and Hochberg (1995)'s[21] procedure: only $p \leq .05 \cdot 5/28 \approx 0.009$ is significant.

[20] We might also be interested in what features of a given sentence, either cross-linguistically in the sense of length, number of predicates, etc., or per-language in the sense of specific morphosyntactic constructions, predict surprisal or cross-linguistic information content of that sentence—but that would of course be an entirely different project with a different setup that would no longer depend on multilinguality at all.

[21] This is a more powerful if conceptually less simple method of correcting for testing multiple hypotheses compared to the more commonly used Bonferroni (1936) correction.

8.5.1 Morphological Counting Complexity

In Chapter 7 we suspected that inflectional morphology was mainly responsible for difficulty in modeling, finding a language's Morphological Counting Complexity (Sagot, 2013) to correlate positively with its difficulty. Using the same MCC values for our 21 Europarl languages, to our surprise, we find no statistically significant correlation with the newly estimated difficulties of our new language models. Comparing the scatterplot for both languages in Fig. 8-13 with Chapter 7's Fig. 7-1, we see that the high-MCC outlier Finnish has become much easier in our (presumably) better-tuned models. Perhaps the reported correlation in that chapter was mainly driven by such outliers and we should conclude that MCC is not a good predictor of modeling difficulty and finer-grained measures of morphological complexity would be more predictive. Such measures might distinguish between the different kinds of axes that make up the table of possible inflections, they make consider phenomena like syncretism, or they may distinguish different *kinds* of inflection (this one we will revisit to some degree in Section 8.5.4).

8.5.2 Head-POS Entropy

Dehouck and Denis (2018) propose an alternative measure of morphosyntactic complexity. Given a corpus of dependency graphs, they estimate the conditional entropy of the POS tag of a random token's parent, conditioned on the token's type. In a language where this **HPE-mean** metric is low, most tokens can predict the POS of their parent even without context or information about word order, which implies that they contain a lot of morphosyntactic information, i.e., the language is likely highly inflected.

We compute HPE-mean from dependency parses of the Europarl data, generated using UDPipe 1.2.0 (Straka et al., 2016) and freely-available tokenization, tagging, parsing models trained on the Universal Dependencies 2.0 treebanks (Straka and Straková, 2017), which leads us to use coarse-grained POS tags that the UD tooling provides.

HPE-mean may be regarded as the mean over all corpus tokens of *Head POS Entropy* (Dehouck and Denis, 2018), which is the entropy of the POS tag of a token's parent given that *particular* token's type. We also compute **HPE-skew**, the (positive) skewness of the empirical distribution of HPE on the corpus tokens. We remark that in each language, HPE is 0 for most tokens.

As predictors of language difficulty, HPE-mean has a Spearman's $\rho = .004/-.045$ ($p > .9/.8$) and HPE-skew has a Spearman's $\rho = .032/.158$ ($p > .8/.4$), so this is not a positive result.

8.5.3 Average dependency length

It has been observed that languages tend to minimize the distance between heads and dependents (Liu, 2008). Speakers prefer shorter dependencies in both production and processing, and average dependency lengths tend to be much shorter than would be expected from randomly-generated parses (Futrell et al., 2015; Liu et al., 2017b). On the other hand, there is substantial variability between languages, and it has been proposed, for example, that head-final languages and case-marking languages tend to have longer dependencies on average.

Do language *models* find short dependencies easier? We again use the automatically-parsed[22] Europarl data and compute dependency lengths using the Futrell et al. (2015) procedure, which excludes punctuation and standardizes several other grammatical relationships (e.g., objects of prepositions are made to depend on their prepositions, and verbs to depend on their complementizers). Our hypothesis that scrambling makes language harder to model seems confirmed at first: while the non-parametric (and thus more weakly powered) Spearman's $\rho = .196/.092$ ($p = .394/.691$), Pearson's $r = .486/.522$ ($p = .032/.015$). However, after correcting for multiple comparisons, this is also non-

[22]In preliminary tests, we find that average dependency lengths estimated from automated parses are very closely correlated with those estimated from (held-out) manual parse trees.

significant.[23]

8.5.4 WALS features

26A (Inflectional Morphology)	BPE	chars
1 Little affixation (5)	-0.0263 (± .034)	0.0131 (± .033)
2 Strongly suffixing (22)	0.0037 (± .049)	-0.0145 (± .049)
3 Weakly suffixing (2)	0.0657 (± .007)	-0.0317 (± .074)
6 Strong prefixing (1)	0.1292	-0.0057
81A (Order of S, O and V)	BPE	chars
1 SOV (7)	0.0125 (± .106)	0.0029 (± .099)
2 SVO (18)	0.0139 (± .058)	-0.0252 (± .053)
3 VSO (5)	-0.0241 (± .041)	-0.0129 (± .089)
4 VOS (2)	0.0233 (± .026)	0.0353 (± .078)
7 No dominant order (4)	0.0252 (± .059)	0.0206 (± .029)

Table 8-II. Average difficulty for languages with certain WALS features (with number of languages), reporting mean and sample standard deviation.

The World Atlas of Language Structures (WALS; Dryer and Haspelmath, 2013) contains nearly 200 binary and integer features for over 2000 languages. Similarly to the Bible situation, not all features are present for all languages—and for some of our Bibles, no information can be found at all. We therefore restrict our attention to two well-annotated WALS features that are present in enough of our Bible languages (foregoing Europarl to keep the analysis simple): 26A "Prefixing vs. Suffixing in Inflectional Morphology" and 81A "Order of Subject, Object and Verb." The results are again not quite as striking as we would hope. In particular, in Mood's median null hypothesis significance test neither 26A ($p > .3$ / .7 for BPE/char-RNNLM) nor 81A ($p > .6$ / .2 for BPE/char-RNNLM) show any significant differences between categories (detailed results in Table 8-II). We therefore turn our attention to much simpler, yet strikingly effective heuristics.

[23]We also caution that the significance test for Pearson's assumes that the two variables are bivariate normal. If not, then even a significant r does not allow us to reject the null hypothesis of zero covariance (Kowalski, 1972, Figs. 1–2, §5).

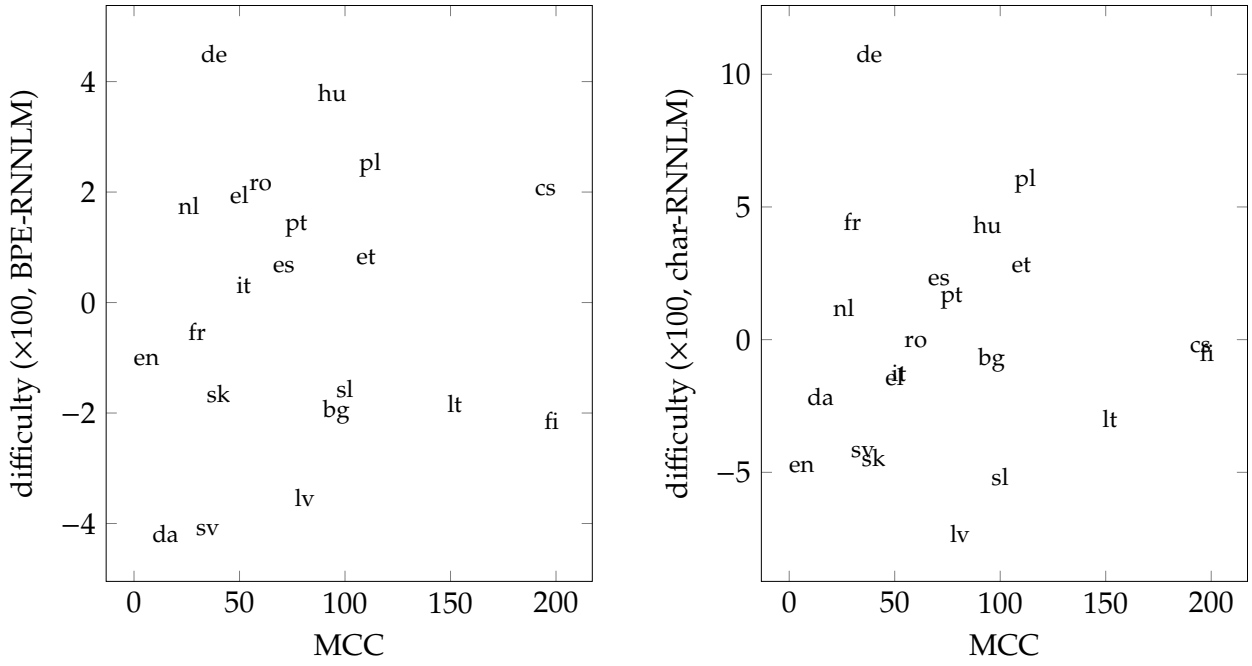


Figure 8-13. MCC does not predict difficulty on Europarl. Spearman's ρ is .091 / .110 with $p > .6$ for BPE-RNNLM (left) / char-RNNLM (right).

8.5.5 Raw character sequence length

		BPE		char	
dataset	statistic	ρ	p	ρ	p
Europarl	Pearson	.509	.0185	**.621**	**.00264**
	Spearman	.423	.0558	**.560**	**.00832**
Bibles	Pearson	.015	.917	**.527**	**.000013**
	Spearman	.014	.915	**.434**	**.000481**

Table 8-III. Correlations and significances when regressing on raw character sequence length. Significant correlations are boldfaced.

An interesting correlation emerges between language difficulty for the char-RNNLM and the raw length in characters of the test corpus (detailed results in Table 8-III). On both Europarl and the more reliable Bible corpus, we have positive correlation for the char-RNNLM at a significance level of $p < .001$, passing the multiple-test correction. The BPE-RNNLM correlation on the Bible corpus is very weak, suggesting that allowing larger units of prediction effectively eliminates this source of difficulty (van Merriënboer et al.,

2017).

8.5.6 Raw word inventory

		BPE		char	
dataset	statistic	ρ	p	ρ	p
Europarl	Pearson	.040	.862	.107	.643
	Spearman	.005	.982	.008	.973
Bibles	Pearson	**.742**	**8e-12**	.034	.792
	Spearman	**.751**	**3e-12**	-.025	.851

Table 8-IV. Correlations and significances when regressing on the size of the raw word inventory.

Our most predictive feature, however, is the *size of the word inventory*. To obtain this number, we count the number of distinct types $|\mathcal{V}|$ in the (pre-tokenized) training set of a language (detailed results in Table 8-IV).[24] While again there is little power in the small set of Europarl languages, on the bigger set of Bibles we do see the biggest positive correlation of any of our features—but only on the BPE model ($p < 1e{-}11$). Recall that the char-RNNLM has no notion of words, whereas the number of BPE units increases with $|\mathcal{V}|$ (indeed, many whole words are BPE units, because we do many merges but BPE stops at word boundaries). Thus, one interpretation is that the Bible corpora are too small to fit the parameters for all the units needed in large-vocabulary languages. A similarly predictive feature on Bibles—whose numerator is this word inventory size—is the type/token ratio, where values closer to 1 are a traditional omen of undertraining.

Unfortunately there may be a confound in this too we have missed: the BPE-RNNLM models use a number of BPE units that is the same across languages *relatively* but not in absolute terms ($0.4 \cdot |\mathcal{V}|$), the word inventory size very much leaks into the model and may drive this correlation.

An interesting observation is that on Europarl, the size of the word inventory and the morphological counting complexity of a language correlate quite well with each other

[24] A more sophisticated version of this feature might consider not just the existence of certain forms but also their rates of appearance. We did calculate the entropy of the unigram distribution over words in a language, but we found that is strongly correlated with the size of the word inventory and not any more predictive.

(Pearson's $\rho = .693$ at $p = .0005$, Spearman's $\rho = .666$ at $p = .0009$), so the original claim in Chapter 7 about MCC may very well hold true after all. Unfortunately, we cannot estimate the MCC for all the Bible languages, or this would be easy to check.[25]

A potentially revealing and either way quite interesting question that is related to that of the word inventory is whether the size of the *lemma* inventory correlates with difficulty. If it does, the issue might be that some languages simply have finer-grained distinctions of meaning that are erased on translation into "easier" languages (or, equivalently, sonewhat arbitrarily decided leading to more information when translating out of the "easier" language that does not distinguish them). If it does not, then perhaps inflectional morphology is strengthened as a candidate for explaining the word inventory correlation. Unfortunately, a study like that will be hard to run for more than the Europarl languages given the reliance on automatic lemmatization tools, and using those would require similar preliminary testing like performed in Section 8.5.3. Given our struggle in finding correlations on the few data points provided by Europarl, we opt to skip this study, leaving an as-large-a-scale-as-possible version with lemma inventories for future work.

8.6 Evaluating Translationese

Our previous experiments treat translated sentences just like natively generated sentences. But since Europarl contains information about which language an intent was originally expressed in,[26] here we have the opportunity to ask another question: is translationese harder, easier, indistinguishable, or impossible to tell?

[25] Perhaps in a future where more data has been annotated by the UniMorph project (Kirov et al., 2018), a yet more comprehensive study can be performed, and the null hypothesis for the MCC can be ruled out after all.

[26] It should be said that using Europarl for translationese studies is not without caveats (Rabinovich et al., 2016), one of them being the fact that not all language pairs are translated equally: a natively Finnish sentence is translated first into English, French, or German (**pivoting**) and only from there into any other language like Bulgarian.

$\mathbf{en}_{\text{native}}$	$\mathbf{en}_{\text{translated}}$	$\mathbf{de}_{\text{native}}$	$\mathbf{de}_{\text{translated}}$	$\mathbf{nl}_{\text{native}}$	$\mathbf{nl}_{\text{translated}}$	...
Resumption...			Wiederauf-...		Hervatten...	...
The German...			Der deutsche...		De Duitse...	...
	Thank you...	Vielen Dank...			Hartelijk...	...
...	...	...	...	...	...	

Figure 8-14. Splitting each language into native and translated leads to missing data.

8.6.1 A flawed attempt

In a first attempt, we may try to tackle this question by simply splitting each language j into two sub-languages, "native" j and "translated" j, resulting in 42 sub-languages with 42 difficulties, with the data now looking something like Fig. 8-14.[27]

Since each intent is expressed in at most 21 sub-languages, this approach *requires* a regresssion method that can handle missing data, such as the probabilistic approach we proposed in Section 8.2. Our mixed-effects modeling ensures that our estimation focuses on the differences between languages, controlling for content by automatically fitting the n_i factors. Thus, we are not in danger of calling native German more complicated than translated German just because German speakers in Parliament may like to talk about complicated things in complicated ways.

Now simply use the already-trained BPE-best models (as they perform the best and are thus most likely to support claims about the language itself rather than the shortcomings of any singular model), limiting ourselves to only splitting the eight languages that have at least 500 native sentences[28] (to ensure stable results).

Indeed we find that native sentences *seem to be* slightly more difficult: as Table 8-V shows, their average d_j is about 0.027 larger for the more powerful BPE-based models (though the arguably less reliable character-level models show the opposite effect even if with smaller magnitude).

[27]This method would also allow us to study the effect of source language, yielding $d_{j \leftarrow j'}$ for sentences translated from j' into j. Similarly, we could have included surprisals from *both* models, *jointly* estimating $d_{j,\text{char-RNN}}$ and $d_{j,\text{BPE}}$ values.

[28]en (3256), fr (1650), de (1275), pt (1077), it (892), es (685), ro (661), pl (594)

language	char-RNN	BPE (0.4$\|\mathcal{V}\|$)	BPE (best)
en	-0.0003	-0.0085	-0.0051
fr	-0.0309	0.0257	0.0285
de	-0.0274	0.0212	0.0226
pt	-0.0174	0.0254	0.0243
it	0.0166	0.0592	0.0607
es	0.0199	0.0652	0.0608
ro	-0.0124	0.0182	0.0201
pl	-0.0774	0.0077	0.0088
average	-0.0162	0.0268	0.0276
sample stdev	0.0310	0.0246	0.0230

Table 8-V. Difference in estimated d_j factors for our three evaluated models

But are they? This result is confounded by the fact that our RNN language models *were trained* mostly on translationese text, by definition (even the English data is mostly translationese, see Fig. 8-7)! Thus, translationese might merely be *different* (Rabinovich and Wintner, 2015)—not necessarily easier to model, but overrepresented when training the model, making the translationese test sentences more predictable.

8.6.2 Controlling for native/translated training data

To remove this confound, we must train our language models on equal parts translationese and native text. We cannot do this for multiple languages at once, given our requirement of training all language models on the same intents. We thus choose to balance only *one* language: train all models for all languages, making sure that the training set for that one language is balanced by removing intents until we have as many intents whose realizations in that language were native as those that were translated from another language. Models thus retrained, we perform the same regression as in Section 8.6.1, but splitting only the chosen language to obtain a native difficulty estimate and a translated one for this language, again reporting the difference between the two.

We sample equal numbers of native and non-native sentences, such that there are ~1M words in the corresponding English column (to be comparable to the PTB size). To raise

language	BPE ($0.4\|\mathcal{V}\|$)
da	0.0281
de	-0.0158
en	-0.0429
es	0.0036
fi	-0.0049
fr	0.0006
it	0.0228
nl	0.0086
pt	-0.0286
sv	-0.0162
average	-0.0045
sample stdev	0.0221

Table 8-VI. Difference in estimated d_j factors when rebalancing training data per language fairly

the number of languages we can split and test in this way, we restrict ourselves here to fully-parallel Europarl in only 10 languages[29] instead of 21, thus ensuring that each of these 10 languages has enough native sentences.

On this level playing field, retraining this time only the model we choose as our favorite (BPE with $0.4 \cdot |\mathcal{V}|$ merges), Table 8-VI shows that the previously observed effect practically disappears (average -0.0045 ± 0.0221 when averaging effects over these ten languages), leading us to question the widespread hypothesis that translationese is "easier" to model (Baker, 1993).[30]

8.7 Conclusion

There is a real danger in cross-linguistic studies of over-extrapolating from limited data. We re-evaluated the conclusions of Chapter 7 on a larger set of languages, requiring new methods to select fully parallel data (Section 8.3.2) or handle missing data. We showed how to fit a paired-sample multiplicative mixed-effects model to probabilistically obtain

[29] da, de, en, es, fi, fr, it, nl, pt, sv

[30] Of course we *cannot* claim that it is just as hard to *read* or *translate* as native text—those are different claims altogether—but only that it is as easy to monolingually language-model.

language difficulties from at-least-pairwise parallel corpora. Our language difficulty estimates were largely stable across datasets and language model architectures, but they were not significantly predicted by linguistic factors. However, a language's vocabulary size and the length in characters of its sentences were well-correlated with difficulty on our large set of languages.

The perhaps biggest takeaway should be that our mixed-effects approach and in particular the focus on using translations as evaluation data with fairly comparable metrics could be used to assess other NLP systems, separating out the influences on performance of language, sentence, model architecture, and training procedure. As a specific example, consider the evaluation of multilingual LLMs. Using translation data as we did here allows us to for example quantify the gap between high-resource and low-resource languages (as represented in training data) so that choices in data, architecture, and training/tuning regimen can be precisely ablated on level playing field. Relaxing the assumption we here made about fully parallel training data means that the data source considered again can make use of the missing data functionality of the mixed-effects model we introduced in this chapter, allowing wide use across languages that may only share some parallel data.

Chapter 9

Measuring Neural Translation Difficulty by Cross-Mutual Information

Published research and text

This chapter is largely taken from Bugliarello et al. (2020).

To round out Part III, we turn our attention to the question of how to measure the difficulty of *translation* using some of the ideas we established in the preceding two chapters.

Machine translation (MT) is one of the core research areas in natural language processing. State-of-the-art MT systems are based on neural networks (Sutskever et al., 2014; Bahdanau et al., 2015), which generally surpass phrase-based systems (Koehn, 2009) in a variety of domains and languages (Bentivogli et al., 2016; Toral and Sánchez-Cartagena, 2017; Castilho et al., 2017; Bojar et al., 2018; Barrault et al., 2019). Various controlled studies to understand where the translation difficulties lie for different language pairs have been conducted for phrase-based MT systems (Birch et al., 2008; Koehn et al., 2009), but this study in 2020 was the first to tackle considering such questions in the context of neural machine translation (NMT). As a result, it is still unclear whether all translation directions are equally easy (or hard) to model for NMT. This chapter hence aims at filling this gap: all else being equal, is it easier to translate from English into Finnish or into Hungarian? And how much easier is it? Conversely, are Finnish and Hungarian equally hard to translate into some other language?

Based on the standard of automated scoring, BLEU (Papineni et al., 2002) scores, previous work (Belinkov et al., 2017) suggests that translating into morphologically rich languages, such as Hungarian or Finnish, is harder than translating into morphologically poor ones (e.g., English). However, a major obstacle in the cross-lingual comparison of MT systems is that many automatic evaluation metrics, including BLEU and METEOR (Banerjee and Lavie, 2005), are *not* in fact cross-lingually comparable, much like (per-word) perplexity and bpc weren't in previous chapters. In fact, being a function of n-gram overlap between candidate and reference translations, they only allow for a fair comparison of the performance between models when translating into the *same* test set in the *same* target language and evaluating against the *same* reference translations. Indeed, one cannot and should not draw conclusions about the difficulty of translating a source language into different target languages purely based on BLEU (or METEOR) scores.

In this chapter, we propose cross-mutual information (XMI): an asymmetric information-theoretic metric of machine translation difficulty that exploits the probabilistic nature of most neural machine translation models. In contrast to BLEU, this information-theoretic quantity no longer explicitly depends on language, model, and pre-tokenization choices. XMI allows us to better evaluate the difficulty of translating text into the target language while controlling for the difficulty of the target-side generation component independent of the translation task.

We perform a case study that, as a starting point, will look only at the task of translating between English and 20 other European languages (as opposed to also between these other languages), making it the first systematic and controlled study of cross-lingual translation difficulties using modern neural translation systems. Code for replicating our experiments is available online at `https://github.com/e-bug/nmt-difficulty`.

Our analysis showcases XMI's potential for shading light on the difficulties of translation as an effect of the properties of the source or target language. We also perform a correlation analysis, in an attempt to further explain our findings. Here, in contrast

to the general wisdom (yet in a way not too shockingly after seeing Chapter 8 fail to reproduce the morphology dependence found in Chapter 7), we find no significant evidence that translating into a morphologically rich language is harder than translating into a morphologically impoverished one. In fact, the only significant correlate of MT difficulty we find is source-side type–token ratio.

9.1 Cross-Linguistic Comparability through Likelihoods, not BLEU

Human evaluation will always be the gold standard of MT evaluation. However, it is both time-consuming and expensive to perform. To help researchers and practitioners quickly deploy and evaluate new systems, automatic metrics that correlate fairly well with human evaluations have been proposed over the years (Banerjee and Lavie, 2005; Snover et al., 2006; Isozaki et al., 2010; Lo, 2019). BLEU (Papineni et al., 2002), however, has remained the most common metric to report the performance of MT systems. BLEU is a precision-based metric: a BLEU score is proportional to the geometric average, for $1 < n < 4$, of the number of (pre-token) n-grams in the candidate translation that also appear in the reference translation.[1]

In the context of our study, we take issue with two shortcomings of BLEU scores that prevent a cross-linguistically comparable study. First, it is not possible to directly compare BLEU scores across languages because different languages might express the same meaning with a very different number of words and thus pre-tokens (assuming standard pre-tokenizers). For instance, agglutinative languages like Turkish often use a single word to express what other languages have periphrastic constructions for. To be concrete, the expression "I will have been programming" is five words in English, but could easily have been one word in a language with sufficient morphological markings; this unfairly

[1]BLEU also corrects for reference coverage and includes a length penalty, but we focus on the high-level picture.

boosts BLEU scores when translating *into* English. The problem is further exacerbated by pre-tokenization techniques as finer granularities result in more partial credit and higher n for the n-gram matches (Post, 2018). In summary, BLEU only allows us to compare architectures and models for a *fixed target language and pre-tokenization scheme*, i.e. it only allows us to draw conclusions about the difficulty of translating different source languages into a specific target one (with downstream performance as a proxy for difficulty). Thus, BLEU scores cannot provide an answer to which translation direction is easier between *any* two source–target language pairs.

In this chapter, we address this particular shortcoming by considering an information-theoretic evaluation. Formally, let $\mathcal{V}_{\mathsf{S}}$ and $\mathcal{V}_{\mathsf{T}}$ be source- and target-language pre-token vocabularies, respectively. Let S and T be source- and target-sentence-valued random variables for languages S and T, respectively; then S and T respectively range over $\mathcal{V}_{\mathsf{S}}^*$ and $\mathcal{V}_{\mathsf{T}}^*$. These random variables S and T are jointly distributed according to some true, unknown probability distribution p.[2] The cross-entropy between the true distribution p and a probabilistic neural translation model $q_{\mathrm{MT}}(T \mid S)$ is defined as:

$$\mathrm{H}_{q_{\mathrm{MT}}}(T \mid S) = -\sum_{\boldsymbol{t} \in \mathcal{V}_{\mathsf{T}}^*} \sum_{\mathbf{s} \in \mathcal{V}_{\mathsf{S}}^*} p(\boldsymbol{t}, \mathbf{s}) \log_2 q_{\mathrm{MT}}(\boldsymbol{t} \mid \mathbf{s}) \tag{9.1}$$

Since we do not know p, we cannot compute Eq. (9.1). However, given a held-out data set of sentence pairs $\{(\mathbf{s}^{(i)}, \boldsymbol{t}^{(i)})\}_{i=1}^{N}$, we can approximate the true cross-entropy as follows:

$$\mathrm{H}_{q_{\mathrm{MT}}}(T \mid S) \approx -\frac{1}{N} \sum_{i=1}^{N} \log_2 q_{\mathrm{MT}}(\boldsymbol{t}^{(i)} \mid \mathbf{s}^{(i)}) \tag{9.2}$$

In the limit as $N \to \infty$, Eq. (9.2) converges to Eq. (9.1) (Cover and Thomas, 2012).

We emphasize using this cross-entropy (that in itself is not novel at all) for evaluation does not rely on language pre-tokenization provided that the model q_{MT} does not, as discussed in Section 2.4 and Chapter 7. While common in the evaluation of language

[2] Admittedly, the process of generating a pair of sentences that happen to be translations does not describe how we usually obtain bi-text: there is one original and then a translation that is made. We should also be careful not to conflate *likely* translations with *good* ones—especially seeing how Section 8.6 claimed that translations are indeed different.

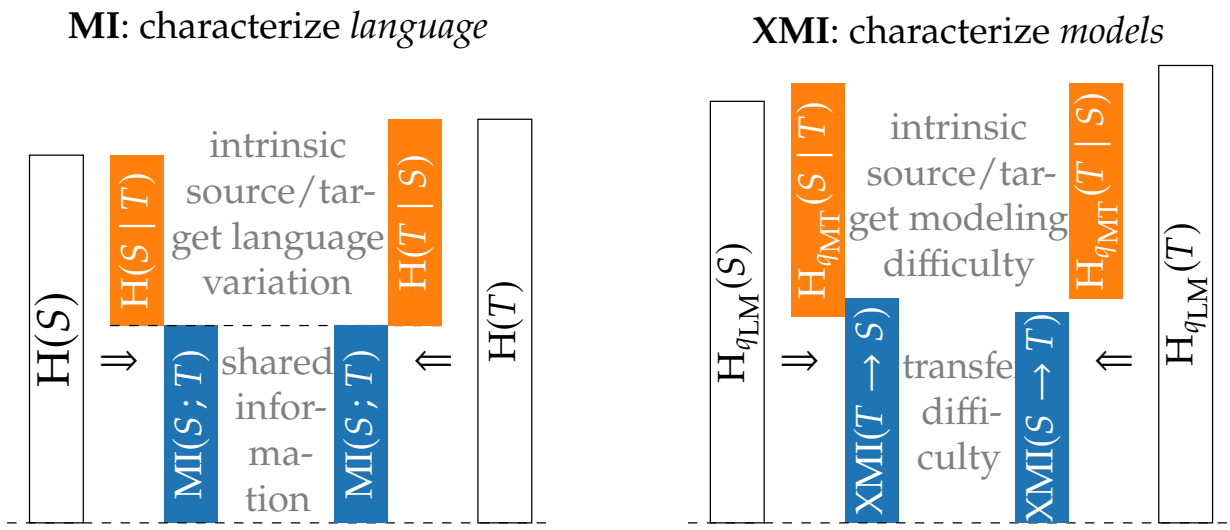


Figure 9-1. Left: Decomposing the uncertainty of a sentence as mutual information plus language-inherent uncertainty: mutual information (MI) corresponds to just how much easier it becomes to predict T when you are given S. MI is symmetric but the relation between $\mathrm{H}(S)$ and $\mathrm{H}(T)$ can be arbitrary. **Right:** estimating cross-entropies using models q_{MT} and q_{LM} invalidates relations between bars, except that $\mathrm{H}_{q_\cdot}(\cdot) \geq \mathrm{H}(\cdot)$. XMI, our proposed metric, is no longer purely a symmetric measure of language, but now an asymmetric measure that mostly highlights models' shortcomings.

models, cross-entropy evaluation has been eschewed in machine translation research since (i) not all MT models are probabilistic (which is a requirement for our method) and (ii) we are often interested in measuring the quality of the candidate translation our model actually produces, e.g. under approximate decoding (the less like sampling decoding looks the more removed our evaluation is from the actual use of the model). However, an information-theoretic evaluation is much more suitable for measuring a more abstract notion of which language pairs are hardest to translate to and from, which is our purpose here.

9.2 Disentangling Translation Difficulty and Monolingual Complexity

We contend that simply reporting cross-entropies is not enough. A second issue in performing a controlled, cross-lingual MT comparison is that the language generation

component (without translation) is not equally difficult across languages, as we established in Chapters 7 and 8. We claim that the difficulty of *translation* corresponds more closely to the **mutual information** $\mathrm{MI}(S;T)$ between the source and target language, which tells us how much easier it becomes to predict T when S is given (see Fig. 9-1). But what is the appropriate analogue of mutual information for cross-entropy, i.e., our case of having two separate models for both terms of the MI? One such natural generalization is a novel quantity that we term **cross-mutual information**, defined as:

$$\mathrm{XMI}(S \to T) = \mathrm{H}_{q_{\mathrm{LM}}}(T) - \mathrm{H}_{q_{\mathrm{MT}}}(T \mid S) \tag{9.3}$$

where $\mathrm{H}(T)$ denotes the entropy of the target sentence T.[3] As in Section 9.1, this can, analogously, be approximated by the cross-entropy of a separate target-side language model q_{LM} over our held-out data set:

$$\mathrm{XMI}(S \to T) \approx -\frac{1}{N} \sum_{i=1}^{N} \log_2 \frac{q_{\mathrm{LM}}(\boldsymbol{t}^{(i)})}{q_{\mathrm{MT}}(\boldsymbol{t}^{(i)} \mid \mathbf{s}^{(i)})} \tag{9.4}$$

which, again, becomes exact as $N \to \infty$. In practice, we note that we mix different distributions $q_{\mathrm{LM}}(\boldsymbol{t})$ and $q_{\mathrm{MT}}(\boldsymbol{t} \mid \mathbf{s})$ and, thus, $q_{\mathrm{LM}}(\boldsymbol{t})$ is *not* necessarily representable by marginalizing $q_{\mathrm{MT}}(\boldsymbol{t} \mid \mathbf{s})$, that is, there need not be any distribution $\tilde{q}(\mathbf{s})$ such that $q_{\mathrm{LM}}(\boldsymbol{t}) = \sum_{\mathbf{s} \in \mathcal{V}_{\mathrm{S}}^*} q_{\mathrm{MT}}(\boldsymbol{t} \mid \mathbf{s}) \cdot \tilde{q}(\mathbf{s})$—XMI might not even be nonnegative in general, unlike mutual information!

While q_{MT} and q_{LM} can, in general, be any two models, we exploit the characteristics of NMT models to provide a more meaningful, model-specific estimate of XMI. NMT architectures typically consist of two components: an encoder that embeds the input text sequence, and a decoder that generates translated output text. The latter acts as a conditional language model, where the source-language sentence embedded by the encoder drives the target-language generation. Hence, we use the architecture of the decoder of q_{MT} as the architecture for our q_{LM} in order to accurately estimate the difficulty of translation for that given architecture.

[3] Unlike in Chapter 7 there is no probabilistic standard metric like bpc that we try to compare and contrast to here, so we don't need to divide by, say, English length to get "XMI per English character."

In summary, by looking at XMI, we can effectively decouple the language generation component (whose difficulties we examined in Chapters 7 and 8) from the translation component. This gives us a measure of how rich and useful is the information extracted in the source language to the target language generation component.

9.3 Experiments

In order to measure which pairs of languages are harder to translate to and from, we make use of Europarl (Koehn, 2005), as in previous chapters, using the latest release $v7$. Note that we will not evaluate *all* hundreds of language pairs, but only the 40 pairs/directions involving English, for multiple reasons. While resource limitations are an obvious reason, we also chose these particular tasks because most of the information available in the web is in English (https://w3techs.com/technologies/overview/content_language) and effectively translating it into any other language would reduce the digital language divide (http://labs.theguardian.com/digital-language-divide/).

Pre-processing steps In order to precisely effect a fully controlled experiment, we again enforce a *fair* comparison by selecting the multi-text set of parallel sentences available across *all* 21 languages in Europarl. This fully controls for the semantic content of the sentences; however, we cannot adequately control for translationese (Stymne, 2017; Zhang and Toral, 2019; Mielke et al., 2019, i.e., our discussion in Section 8.6). The number of parallel sentences available in Europarl varies considerably, ranging from 387K sentences for Polish-English to 2.3M sentences for Dutch-English. Finding a shared core as in Chapter 7 and Chapter 8 leaves us with 197,919 intents for every language pair we consider, from which we then extract 1,000 and 2,000 unique, random intents for validation and test, respectively.

Next, we follow the same pre-processing steps used by Vaswani et al. (2017) to train the Transformer model on WMT data: Data sets are first pre-tokenized using the Moses toolkit (Koehn et al., 2007) and then filtered by removing sentences longer than 80 tokens in

either source or target language. Due to this cleaning step that is specific to each training corpus, different sentences are dropped in each data set. We then only select the set of sentence pairs that are shared across all languages. This results in a final set of 190,733 training intents.

For each parallel corpus, we jointly learn byte-pair encodings (BPE; Sennrich et al., 2016) for the source and target languages, using 16,000 merge operations. We use the same vocabularies for the language models.[4]

Setup In our experiments, we train 1-1 Transformer models (Vaswani et al., 2017), which often achieve state-of-the-art performance on MT for various language pairs. In particular, we rely on the PyTorch (Paszke et al., 2019) re-implementation of the Transformer model in the Fairseq toolkit (Ott et al., 2019).

All MT experiments are based on the Base Transformer architecture, which we trained for 20,000 steps and evaluated using the checkpoint corresponding to the lowest validation loss. We trained our models on a cluster of 4 machines, each equipped with 4 Nvidia P100 GPUs, resulting in training times of almost 70 minutes for each system. Sentence pairs with similar sequence length were batched together, with each batch containing a total of approximately $32K$ source tokens and $32K$ target tokens. We used the hyper-parameters specified in latest version (3) of Google's Tensor2Tensor (Vaswani et al., 2018) implementation, with the exception of the dropout rate, as we found 0.3 to be more robust across all the models trained on Europarl. Models are optimized using Adam (Kingma and Ba, 2015) and following the learning schedule specified by Vaswani et al. (2017) with 8,000 warm-up steps. We employed label smoothing[5] $\epsilon_{ls} = 0.1$ (Szegedy et al., 2016) during training, given that it has been shown to prevent models from over-confident predictions, which helps to regularize the models. We use beam search with a beam size of 4 and length penalty $\alpha = 0.6$ (Wu et al., 2016).

[4]For English, we arbitrarily chose the English portion of the `en-bg` vocabulary.
[5]The "labels" in question are simply what is being predicted at every prediction step: tokens.

For language modeling, we use a decoder using the same architecture and hyperparameters, training it at the sentence level, as opposed to commonly used fixed-length chunks. We train our language models, again using label smoothing $\epsilon_{ls} = 0.1$. Training progresses for 10,000 steps on sequences consisting of separate sentences in our corpus. Analogously to translation models, the checkpoints corresponding to the lowest validation losses were used for evaluation.

We report cross-entropies ($H_{q_{MT}}$, $H_{q_{LM}}$), XMI, and BLEU scores obtained using SacreBLEU (Post, 2018).[6] Finally, as explained and motivated in Section 2.4 and Chapter 7, we multiply per-token cross-entropy values by the number of tokens generated by each model to make our quantities independent of sentence lengths (and divide them by the total number of sentences to be in line with our motivation and definitions above, which as with dividing by English characters in Chapter 7 does not change results).

9.4 Results and Analysis

We train 40 systems, translating each language into and from English. The models' performance in terms of BLEU scores, and the cross-mutual information (XMI) and cross-entropy values over the test sets are reported in Table 9-I.

We also tested significance and robustness of results by applying bootstrap re-sampling (Koehn, 2004) on either training or test sets to the systems achieving the highest and the lowest BLEU scores in the validation set for each direction. Table 9-II presents the results of these experiments; we observe a general trend where the performance of different models varies similarly. For instance, when we bootstrap test sets, we see that the average BLEU scores are equal to the ones seen in Table 9-I, and that all the models have similar confidence intervals.[7] When bootstrapping the training data, we observe a consistent drop in mean performance of $2 - 3$ BLEU points across the translation tasks. The drop in performance

[6]Signature: BLEU+c.mixed+#.1+s.exp+tok.13a+v.1.2.12.
[7]The same results were observed in all of the 40 models.

◌ → **en**	bg	cs	da	de	el	**es**	et	fi	fr	hu
BLEU	47.4	42.4	46.3	44.0	50.0	50.6	39.3	38.2	44.9	38.4
XMI(◌ → en)	102.3	97.0	99.7	96.5	105.3	103.8	92.8	92.1	97.0	92.5
$H_{q_{LM}}$(en)	154.2									
$H_{q_{MT}}$(en | ◌)	51.8	57.2	54.5	57.7	48.9	50.4	61.4	62.0	57.2	61.6
en → ◌	bg	cs	da	de	el	es	et	fi	fr	hu
BLEU	46.3	34.7	45.0	36.3	45.5	50.2	27.7	30.5	45.7	30.3
XMI(en *to* ◌)	106.2	102.8	103.3	104.0	111.0	108.1	100.2	98.0	99.7	99.1
$H_{q_{LM}}$(◌)	156.5	164.0	152.7	167.6	163.7	159.3	162.5	158.6	154.9	166.6
$H_{q_{MT}}$(◌ | en)	50.3	61.2	49.4	63.6	52.7	51.3	62.4	60.6	55.1	67.5

◌ → **en**	it	**lt**	lv	nl	pl	pt	ro	sk	sl	sv	**avg**
BLEU	40.8	37.6	40.3	38.3	39.8	48.3	50.5	44.2	45.3	43.7	43.5
XMI(◌ → en)	92.1	89.2	94.2	86.5	91.9	102.5	106.1	99.8	100.1	96.9	96.9
$H_{q_{LM}}$(en)	154.2										154.2
$H_{q_{MT}}$(en | ◌)	62.1	65.0	60.0	67.7	62.3	51.7	48.1	54.4	54.1	57.3	57.3
en → ◌	it	lt	lv	nl	pl	pt	ro	sk	sl	sv	**avg**
BLEU	37.9	31.0	34.6	34.9	30.5	46.7	44.2	39.8	41.5	41.3	38.73
XMI(en *to* ◌)	95.3	96.0	99.3	90.4	98.3	105.2	112.4	105.8	107.9	100.1	102.1
$H_{q_{LM}}$(◌)	158.6	159.2	156.4	159.7	163.4	159.3	160.5	157.7	158.2	153.1	159.6
$H_{q_{MT}}$(◌ | en)	63.3	63.1	57.0	69.3	65.1	54.1	48.1	51.9	50.3	53.0	57.5

Table 9-I. Test scores, from and into English, Europarl, visualized in Fig. 9-2 and Fig. 9-3. Boldfaced Lithuanian (lt) and Spanish (es) are easiest and hardest to translate out of, respectively (Section 9.4.1); the shaded languages and cells are referred to in Section 9.4.2.

is not surprising as the resulting training sets are more redundant, having fewer unique sentences than the original sets, but it is interesting to see that all models are similarly affected. The standard deviation over 5 runs is also similar across all models but slightly larger on the high-performing ones.

9.4.1 Translating into English

When translating into the same target language (in this case, English), BLEU scores are, in fact, comparable, and can be used as a proxy for difficulty. We can then conclude, for instance, that Lithuanian (`lt`) is the hardest language to translate from, while Spanish (`es`)

Model	Train bootstrap	Test bootstrap
en-es	47.6 (0.233)	50.2 (0.026)
en-et	25.6 (0.167)	27.7 (0.026)
lt-en	34.5 (0.150)	37.6 (0.027)
ro-en	47.5 (0.232)	50.5 (0.027)

Table 9-II. Mean test BLEU scores when bootstrapping train and test sets. Numbers in brackets denote standard deviation over 5 runs (train bootstrap) and 95% confidence interval over $1,000$ samples (test bootstrap). Bootstrapping training data by definition reduces the available training data for any given run, lowering performance.

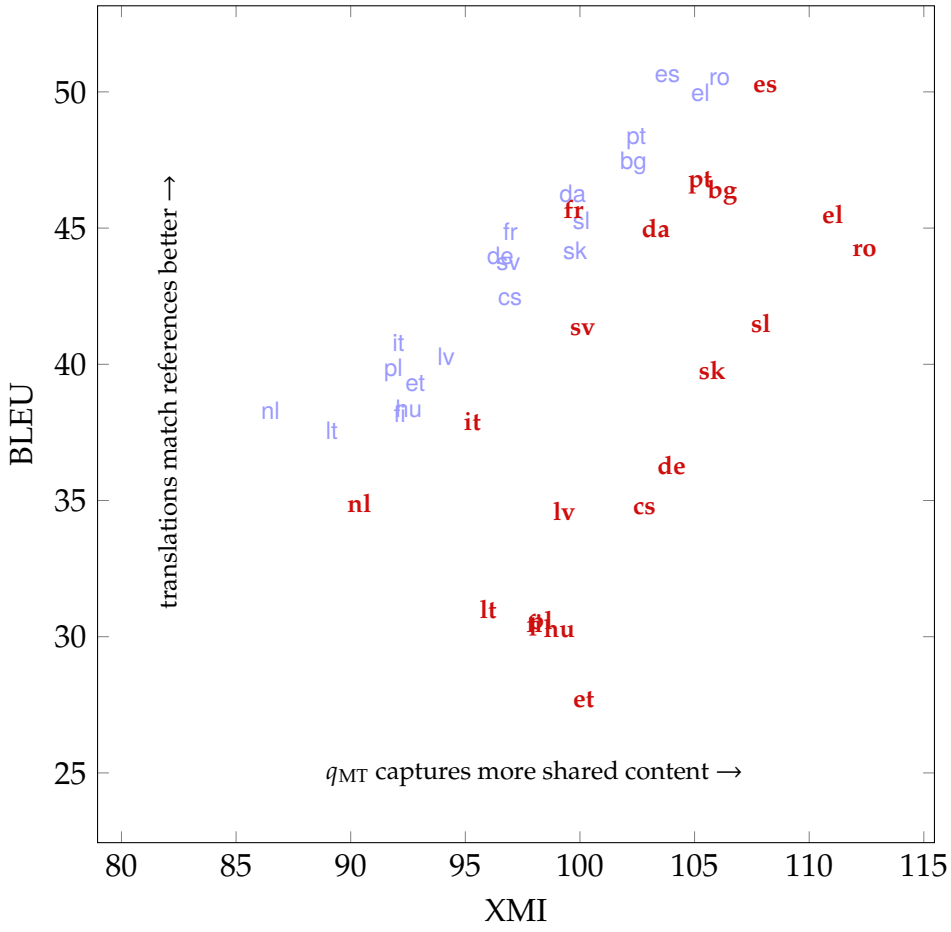


Figure 9-2. Correlation between XMI and BLEU from Table 9-I, into and **from** English. More correlations in Fig. 9-4.

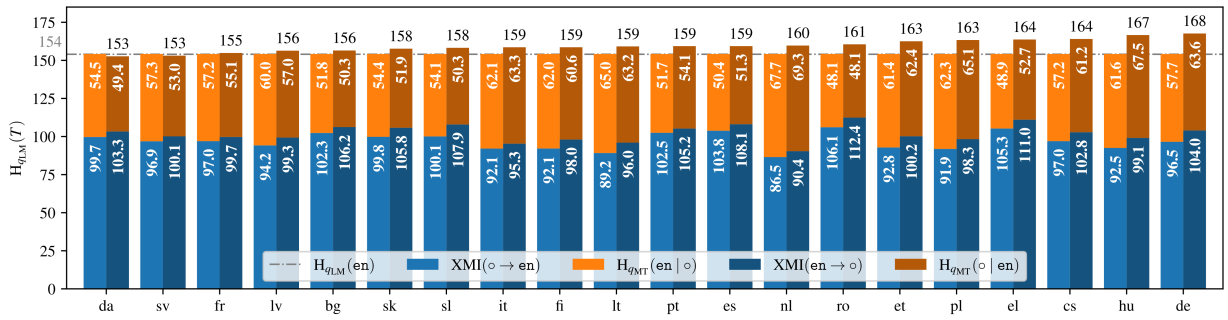


Figure 9-3. $\mathrm{H}_{q_{\mathrm{LM}}}(T)$, decomposed into $\mathrm{XMI}(S \rightarrow T)$, the information that the system successfully transfers, and $\mathrm{H}_{q_{\mathrm{MT}}}(T \mid S)$, the uncertainty that remains in the target language, all measured in bits. Note that in $\mathrm{XMI}(S \rightarrow T)$ the translation is from the left to the right argument.

is the easiest. In this scenario, given the good correlation of BLEU scores with human evaluations, it is desirable that XMI correlates well with BLEU. This behavior is indeed apparent in the blue points in Fig. 9-2, confirming the efficacy of XMI in evaluating the difficulty of translation while still being independent of the target language generation component.

9.4.2 Translating from English

Despite the large gaps between BLEU scores in Table 9-I, one should not be tempted to claim that it is easier to translate into English than from English for these languages as often hinted at in previous work (e.g., Belinkov et al., 2017). As we described above, BLEU scores in different target languages are not directly comparable, and we actually find that XMI is slightly higher, on average, when translating from English, indicating that it is actually *easier*, on average, to transfer information correctly in this direction. For instance, translation from English to Finnish is shown to be easier (in this specific sense with this specific metric) than from Finnish to English (98.0 vs. 92.1), despite the large gap in the opposite direction in BLEU scores (30.5 vs. 38.2; in both metrics "higher is better"). This suggests that the former model is heavily penalized by the target-side language model; this is likely because Finnish has a large number of inflectional types for nouns and verbs. Another interesting example is given by Greek (`el`) and Spanish (`es`) in Table 9-I, where,

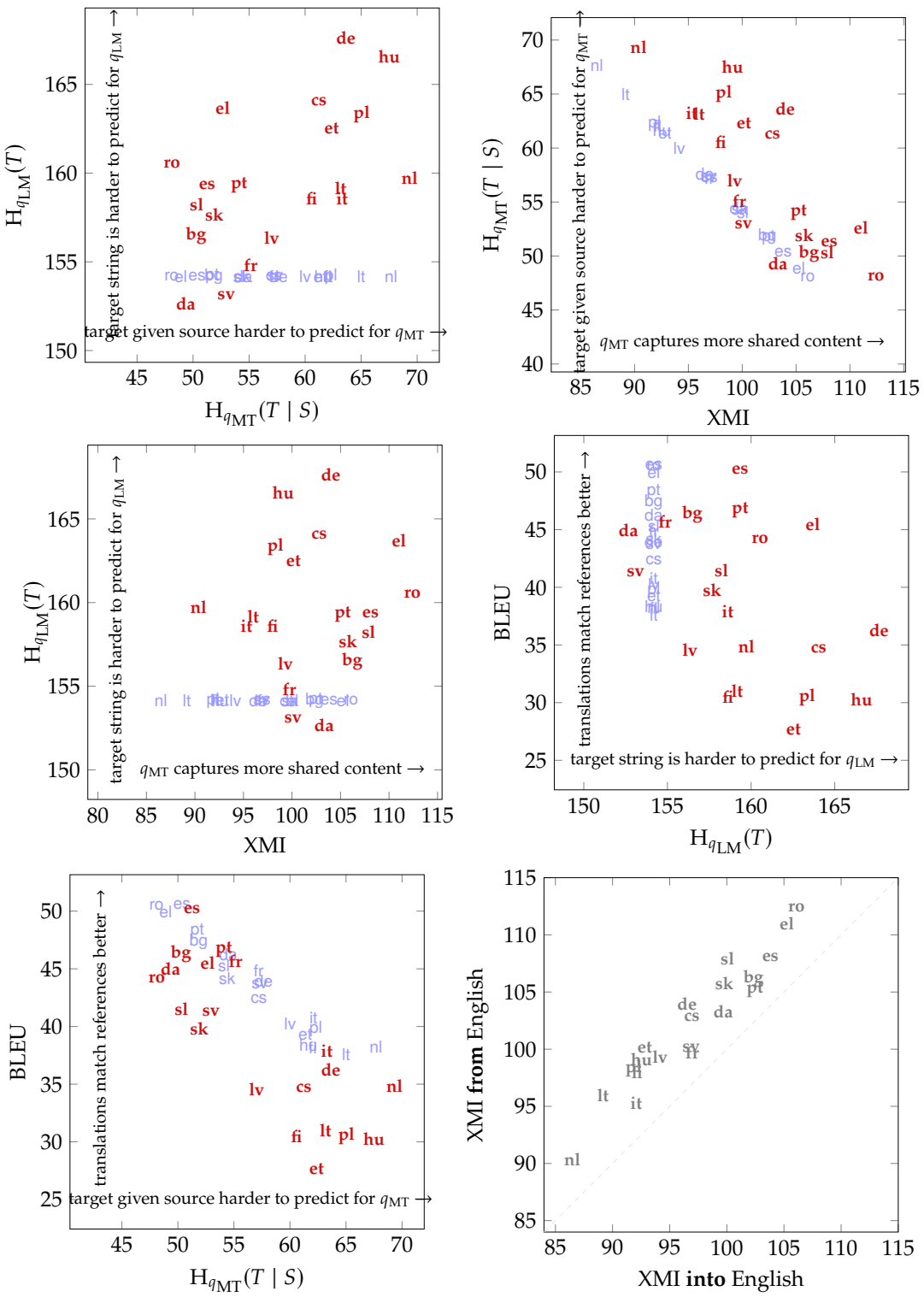


Figure 9-4. More correlations between metrics in Table 9-I, into and **from** English.

again, the two tasks achieve very different BLEU scores but similar XMI. In light of the correlation with BLEU when translating into English, this shows us that Greek is just harder to language model, corroborating the findings of Chapter 8. Moreover, Fig. 9-2 clearly shows that, as expected, XMI is not as well correlated with BLEU when translating from English, given that BLEU scores are not cross-lingually comparable.

Figure 9-4 shows more correlations between the metrics we reported in our experiments (see Table 9-I).

9.4.3 Correlations with linguistic and data features

Last, we conducted a correlation study between the translation difficulties as measured by XMI and the linguistic and data-dependent properties of each translation task, following the approaches of Lin et al. (2019) and Section 8.5 and evaluating:

- MCC: Morphological counting complexity (Sagot, 2013), using the values for Europarl used in Chapters 7 and 8.
- ADL: Average dependency length (Futrell et al., 2015), using the values obtained for Europarl in Section 8.5.
- HPE-mean: mean over all Europarl tokens of Head-POS Entropy (Dehouck and Denis, 2018), as used before in Section 8.5.
- Six different linguistic distances (genetic, syntactic, featural, phonological, inventory, geographic) from the URIEL Typological Database (Littell et al., 2017). We refer the reader to Lin et al. (2019) for more details.
- Word number ratio: number of source tokens over number of target tokens used for training.
- TTR_{src} and TTR_{tgt}: type-to-token ratio evaluated on the source and target language training data, respectively, to measure lexical diversity.
- d_{TTR}: distance between the TTRs of the source and target language corpora, as a

Metric	Pearson			Spearman		
	◌→ en	en → ◌	both	◌→ en	en → ◌	both
MCC_{src}	-0.2579 (0.2723)	–	-0.4302 (0.0056)	-0.2135 (0.3660)	–	-0.4444 (0.0041)
MCC_{tgt}	–	-0.1260 (0.5965)	0.2619 (0.1025)	–	-0.1263 (0.5957)	0.3778 (0.0162)
ADL_{src}	-0.2972 (0.2032)	–	-0.1166 (0.4737)	-0.2887 (0.2170)	–	0.0166 (0.9188)
ADL_{tgt}	–	-0.2254 (0.3393)	-0.2110 (0.1912)	–	-0.1820 (0.4426)	-0.3798 (0.0156)
HPE-$mean_{src}$	0.2012 (0.3950)	–	0.4567 (0.0031)	0.2000 (0.3979)	–	0.4508 (0.0035)
HPE-$mean_{tgt}$	–	0.0142 (0.9525)	-0.4115 (0.0083)	–	0.0120 (0.9599)	-0.4103 (0.0085)
genetic	0.0433 (0.8563)	0.0777 (0.7446)	0.0544 (0.7387)	-0.1526 (0.5207)	-0.1741 (0.4630)	-0.1360 (0.4028)
syntactic	-0.3643 (0.1143)	-0.2056 (0.3845)	-0.2556 (0.1114)	-0.3560 (0.1234)	-0.2695 (0.2506)	-0.2688 (0.0935)
featural	-0.0561 (0.8142)	-0.0577 (0.8090)	-0.0511 (0.7540)	0.0121 (0.9597)	-0.0093 (0.9690)	-0.0109 (0.9467)
phonological	-0.1442 (0.5441)	-0.2222 (0.3465)	-0.1647 (0.3097)	-0.0435 (0.8556)	-0.0948 (0.6909)	-0.0906 (0.5782)
inventory	0.1125 (0.6369)	0.1048 (0.6601)	0.0976 (0.5492)	0.1231 (0.6052)	0.1472 (0.5356)	0.1128 (0.4884)
geographic	0.1983 (0.4019)	0.3388 (0.1440)	0.2416 (0.1332)	0.1336 (0.5745)	0.2550 (0.2779)	0.2062 (0.2017)
word number ratio	0.4559 (0.0434)	-0.2953 (0.2063)	0.2988 (0.0611)	0.4602 (0.0412)	-0.3278 (0.1582)	0.3570 (0.0237)
TTR_{src}	-0.4746 (0.0345)	–	**-0.5196 (0.0006)**	-0.4857 (0.0299)	–	**-0.5136 (0.0007)**
TTR_{tgt}	–	-0.2931 (0.2099)	0.1651 (0.3086)	–	-0.3128 (0.1794)	0.3355 (0.0343)
d_{TTR}	-0.4434 (0.0502)	-0.2404 (0.3072)	-0.4427 (0.0042)	-0.4857 (0.0299)	-0.3128 (0.1794)	**-0.4660 (0.0024)**
word overlap ratio	0.2563 (0.2754)	0.0526 (0.8258)	0.1383 (0.3949)	0.1474 (0.5352)	0.1474 (0.5352)	0.1731 (0.2853)

Table 9-III. All Pearson's and Spearman's correlation coefficients and corresponding p-values (in brackets) between XMI and various metrics. Values in black are statistically significant in isolation at $p < 0.05$, and bold values are also statistically significant after Bonferroni correction (uncorrected $p < 0.0029$).

rough indication of their morphological similarity:

$$d_{\mathrm{TTR}} = \left(1 - \frac{\mathrm{TTR}_{src}}{\mathrm{TTR}_{tgt}}\right)^2.$$

- Word overlap ratio: we measure the similarity between the vocabularies of source and target languages as the ratio between the number of shared types and the size of their union.

Table 9-III lists Pearson's and Spearman's correlation coefficients for data-dependent metrics, where bold values indicate statistically significant results ($p < 0.05$) after Bonferroni correction ($p < 0.0029$). Results are given both for only translations into English and out of English as well as the aggregate of both, which we primarily look at for results.

Interestingly, the only features that significantly correlate with our metric are related to the type-to-token ratio (TTR) for the source language and the distance between source and target TTRs. This implies that a potential explanation for the differences in translation difficulty lies in lexical variation.

9.5 Conclusion

In this final chapter, we proposed a novel information-theoretic approach, XMI, to measure the *translation* difficulty of probabilistic MT models. Differently from BLEU and other metrics, ours is again language- and pre-tokenization-agnostic, enabling the first systematic and controlled study of cross-lingual translation difficulties. Our results show that XMI correlates well with BLEU scores when translating into the same language (where they are comparable), and that higher BLEU scores in different languages do not necessarily imply easier translations.

Chapter 10

Conclusion

While this thesis connected a variety of different contributions and ideas, we hope that that fact serves as illustration that language model research does not need to revolve around data and compute moats and most impressive generation demos. There are and remain many questions and opportunities on both construction and evaluation as well as creative use of open-vocabulary language models.

10.1 What did we achieve?

We contributed both to the construction of open-vocabulary language models as well as to their evaluation in fair cross-linguistic comparisons, even showing how to extend that line of thinking to bilingual translation models.

In the realm of construction, we began by providing a fairly comprehensive survey of tokenization efforts past and present, illuminating background and connections (Chapter 3). With this knowledge, we contributed a very simple, yet clean and powerful pre-tokenizer that allows for word-like unit pre-tokenization while being as reversible as makes sense given the Python landscape, a quality usually underappreciated in tokenization, but one that is crucial for fair evaluation (Section 4.1.2). To prepare us for construction efforts that are not tied to a specific autoregressive modeling skeleton, we then provided a unifying view on RNNs and Transformers for the purpose of neural autoregressive language

modeling (Section 4.2.2). Using this framework, we proposed a quite ambitious theoretical open-vocabulary neural language model that is based on linguistic and modeling desiderata that are not met in existing models, but given that its demand of an infinite vocabulary of latent lexemes through the use of Bayesian nonparametrics proved too much for this thesis, we only sketched inference ideas that future work may very well find useful to make this lofty dream of a truly infinite vocabulary a reality (Chapter 5). Not content to stop there, we developed a finite vocabulary simplification that is based on the same desiderata, but by giving up on infinite latent lexemes and instead using a finite vocabulary and a mechanism of controlled back-off to a character-level model, is much more feasible to implement and train, reaching state-of-the-art at publication time (Chapter 6). Only a side note in this evaluation, we proposed what at the time was an uncommon idea: use BPE for language modeling, showing that it is a strong baseline that beats existing state-of-the-art models and sometimes even our own models at the time of publication (Sections 4.2.3 and 6.4).

In the realm of evaluation, we began with a simple question: are some languages harder to model than others? And if so, is there some predictability to this difficulty? (Chapter 7). To answer this question, we posited the necessity of three ingredients: open-vocabulary language models (as UNKs lead to "cheating" in probabilistic evaluation), the use of translations, that is, of multi-text, for training and evaluation, and the use of a language-independent metric, which we found in the form of total surprisal, either normalized to bits per English character (bpec) to make it intuitively comparable to bits per character (bpc) of a character-level model (Chapter 7) or as a raw score itself (Chapter 8). We performed a pilot study on 21 European languages, finding that indeed there are noticeable differences in performance and at least in this pilot study it seems like more complex inflectional morphology may correlate with such difficulty, a claim strengthened by an intervention of lemmatization. Following this exploratory pilot, we devised a far bigger study that would scale up to 69 languages, using an ILP formulation to optimize for maximum of Bibles alongside the European language corpus, and a family of more

sophisticated mixed effects models to calculate and compare language difficulty for given language models that allows for missing data at test time (Chapter 8). This bigger study then contained three experiments. First, we repeated the pilot experiments now using the best-fitting model of this family to compare language models' difficulties with our 69 languages, evaluating different RNNLM types with different granularities in characters or different BPE settings, and different Bible translations in the same language (Section 8.4). Second, we more thoroughly investigated the correlation between difficulty and linguistic as well as data-driven properties of languages, finding that the only significant correlations are with simple metrics of the data (Section 8.5). Third, we used the ability to deal with missing data at test time to produce new insights in the relative difficulty of Translationese, showing it is different, but no easier to model (Section 8.6). Building on the successes of these approaches, we showed that the methods we have established for monolingual language models can be of great use in analyzing bilingual translation models as well (provided they are probabilistic generative models), introducing a new metric to tease apart the monolingual generation aspect of translation from the bilingual extraction and transformation aspect (Chapter 9).

Through all this, we have highlighted achievements and contributions, but in these final pages it may be worth pointing out shortcomings and open questions as well.

10.2 What now?

As teased in Chapter 1, much of this work was performed using a generation of language models that has fallen out of favor and while the inquiry into their working still has use and scientific merit, the claim that what we built could be transfered to more modern models, while theoretically true, still needs to be empirically tested. Perhaps some of the tricks we used to make our finite-vocabulary model from Chapter 6 work are no longer necessary in a bigger model or one based on Transformers, and maybe different tricks would be needed

to ensure convincing performance. Creative adaption of the Transformer skeleton and its ability to choose positional embeddings quite flexibly rather than relying on the strict sequence inherent in RNNs may even lead to entire different ways of implementing our ideas that may make the INLM of Chapter 5 far easier to implement or approximate? Even if that model stays the same, successful INLM inference on real data will likely not use the sketch we provided to show that implementation should be theoretically possible. Most likely, clever approximation and sampling procedures will need to be devised to scale an infinite-vocabulary model to real data, especially on a large scale.

It is worth discussing that the BPE baseline we provided with our results in Chapter 6 may look disheartening for research that simply wants to create a well-working general-purpose language model. Indeed, the fact that we did not end up using our finite-vocabulary model in Part III of this thesis proves that we, too, preferred a conceptually simpler model that performed comparably. Again, in the game of the lowest numbers through the biggest models, what we provide here may not be particularly useful, but we must remember that there is more to language modeling and NLP as a whole than leaderboard-climbing on English (or maybe a handful of related languages).

This leads us to Part III where we decided to put existing language models to the test and find out just how language-independent they truly are. The quite significant differences we found in Chapter 7 were a success in the sense that truly we had found something interesting worth studying, but the fact that we failed to replicate the effect of morphology on difficulty was a very confusing setback. Along with the notable variance between different Bible translations in Section 8.4.2, this calls our evaluation method into question. Indeed, this kind of inquiry is fraught with so many open questions and so much uncertainty about claims like "translations preserve information and not do not add any new information" such that even though the question may be well-formed, in theory, any practical answer will likely have to be questioned in some way or other. The fact that data-driven statistics were so much more predictive than any linguistic categories or values

on the other hand was a result that at first looks like a disappointing setback, but in fact is quite interesting in that it shows us that language models still have a long way to go to not be bogged down by likely very simple matters of data flow.

Finally, our extension to translation in Chapter 9 is an interesting case in that the metric we propose is one that is likely of no direct interest to any practitioner, who very much should stay interested in task-oriented metrics like BLEU. Rather than attempting to supplant these metrics to be all the rage on the leaderboards, again, we are more interested in stimulating thought about matters of translation itself and what it really consists of.

That, perhaps, is our biggest hope for this thesis: that it will stimulate thought and discussion about the many open questions in language modeling and NLP as a whole, that it will make readers reconsider problems they thought they had long fully understood, and that it will help lift the blinders of a researcher like the very author of this thesis to lead them to an open-minded position of being comfortable with knowing that they don't really know what's going on—and that there are many exciting little discoveries to be made along every step of the way.

Bibliography

Željko Agić, Anders Johannsen, Barbara Plank, Héctor Martínez Alonso, Natalie Schluter, and Anders Søgaard. 2016. Multilingual projection for parsing truly low-resource languages. *Transactions of the Association for Computational Linguistics*, 4:301–312.

Gustavo Aguilar, Bryan McCann, Tong Niu, Nazneen Rajani, Nitish Keskar, and Thamar Solorio. 2021. Char2subword: Extending the subword embedding space using robust character compositionality.

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2019. Character-level language modeling with deeper self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3159–3166.

David Aldón Mínguez, Marta Ruiz Costa-Jussà, and José Adrián Rodríguez Fonollosa. 2016. Neural machine translation using bitmap fonts. In *Proceedings of the EAMT 2016 Fifth Workshop on Hybrid Approaches to Translation (HyTra)*, pages 1–9.

Chantal Amrhein and Rico Sennrich. 2021. How suitable are subword segmentation strategies for translating non-concatenative morphology? In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 689–705, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nicholas Andrews, Mark Dredze, Benjamin Van Durme, and Jason Eisner. 2017. Bayesian modeling of lexical resources for low-resource settings. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 1029–1039, Vancouver.

Duygu Ataman, Wilker Aziz, and Alexandra Birch. 2020. A latent morphology model for open-vocabulary neural machine translation. In *International Conference on Learning Representations*.

Duygu Ataman and Marcello Federico. 2018. Compositional representation of morphologically-rich input for neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 305–311, Melbourne, Australia. Association for Computational Linguistics.

Duygu Ataman, Orhan Firat, Mattia A. Di Gangi, Marcello Federico, and Alexandra Birch. 2019. On the importance of word boundaries in character-level neural machine translation. In *Proceedings of the 3rd Workshop on Neural Generation and Translation*, pages 187–193, Hong Kong. Association for Computational Linguistics.

Harald Baayen and Richard Sproat. 1996. Estimating lexical priors for low-frequency syncretic forms. *Computational Linguistics*, 22(2):155–166.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the 3rd International Conference on Learning Representations*.

Mona Baker. 1993. Corpus linguistics and translation studies: Implications and applications. *Text and Technology: In Honour of John Sinclair*, 233:250.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Loïc Barrault, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Matthias Huck, Philipp Koehn, Shervin Malmasi, Christof

Monz, Mathias Müller, Santanu Pal, Matt Post, and Marcos Zampieri. 2019. Findings of the 2019 Conference on Machine Translation (WMT19). In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 1–61, Florence, Italy. Association for Computational Linguistics.

Matthew J. Beal, Zoubin Ghahramani, and Carl Edward Rasmussen. 2001. The infinite hidden markov model. In *NIPS*, pages 577–584.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do Neural Machine Translation Models Learn about Morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, Vancouver, Canada. Association for Computational Linguistics.

Emily M. Bender. 2009. Linguistically naïve != language independent: Why NLP needs linguistic typology. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 26–32, Athens, Greece. Association for Computational Linguistics.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2001. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432.

Yoav Benjamini and Yosef Hochberg. 1995. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):289–300.

Luisa Bentivogli, Arianna Bisazza, Mauro Cettolo, and Marcello Federico. 2016. Neural versus Phrase-Based Machine Translation Quality: a Case Study. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 257–267, Austin, Texas. Association for Computational Linguistics.

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 582–590, Los Angeles, California. Association for Computational Linguistics.

Parminder Bhatia, Robert Guthrie, and Jacob Eisenstein. 2016. Morphological priors for probabilistic neural word embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 490–500, Austin, Texas. Association for Computational Linguistics.

Alexandra Birch, Miles Osborne, and Philipp Koehn. 2008. Predicting Success in Machine Translation. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 745–754, Honolulu, Hawaii. Association for Computational Linguistics.

Maximilian Bisani and Hermann Ney. 2005. Open vocabulary speech recognition with flat hybrid models. In *INTERSPEECH*, pages 725–728.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5(0):135–146.

Ondřej Bojar, Christian Federmann, Mark Fishel, Yvette Graham, Barry Haddow, Philipp Koehn, and Christof Monz. 2018. Findings of the 2018 Conference on Machine Translation (WMT18). In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 272–303, Belgium, Brussels. Association for Computational Linguistics.

Carlo Bonferroni. 1936. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62.

Samuel Broscheit. 2018. Learning distributional token representations from visual features.

In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 187–194, Melbourne, Australia. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Jacob Buckman and Graham Neubig. 2018. Neural lattice language models. *Transactions of the Association for Computational Linguistics (TACL)*, 6.

Emanuele Bugliarello, Sabrina J. Mielke, Antonios Anastasopoulos, Ryan Cotterell, and Naoaki Okazaki. 2020. It's easier to translate out of English than into it: Measuring neural translation difficulty by cross-mutual information. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1640–1649, Online. Association for Computational Linguistics.

Kris Cao and Laura Rimell. 2021. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bob Carpenter, Andrew Gelman, Matthew Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of Statistical Software, Articles*, 76(1):1–32.

Sheila Castilho, Joss Moorkens, Federico Gaspari, Iacer Calixto, John Tinsley, and Andy

Way. 2017. Is Neural Machine Translation the New State of the Art? *The Prague Bulletin of Mathematical Linguistics*, 108(1):109–120.

William Chan, Yu Zhang, Quoc Le, and Navdeep Jaitly. 2017. Latent sequence decompositions. In *International Conference on Learning Representations 2017*.

Stanley F. Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

Colin Cherry, George Foster, Ankur Bapna, Orhan Firat, and Wolfgang Macherey. 2018. Revisiting character-based neural machine translation with capacity and compression. In *EMNLP*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Dokook Choe, Rami Al-Rfou, Mandy Guo, Heeyoung Lee, and Noah Constant. 2019. Bridging the gap for tokenizer-free language models.

Grzegorz Chrupała. 2013. Text segmentation with character-level text embeddings. Workshop on Deep Learning for Audio, Speech and Language Processing, ICML 2013; Workshop on Deep Learning for Audio, Speech and Language Processing, ICML 2013 ; Conference date: 16-06-2013.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *International Conference on Learning Representations 2017*.

Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. 2016. A character-level de-

coder without explicit segmentation for neural machine translation. *arXiv preprint arXiv:1603.06147*.

Kenneth W. Church. 2000. Empirical estimates of adaptation: The chance of two Noriegas is closer to $p/2$ than p^2. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 1*, pages 180–186. Association for Computational Linguistics.

Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. 2021. Canine: Pre-training an efficient tokenization-free encoder for language representation. *arXiv preprint arXiv:2103.06874*.

Lionel Clément, Eric De la Clergerie, and Lionel Net. 2005. Maf: a morphosyntactic annotation framework.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2017. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1107–1116, Valencia, Spain. Association for Computational Linguistics.

Marta R. Costa-jussà, David Aldón, and José A. R. Fonollosa. 2017. Chinese–spanish neural machine translation enhanced with character and word bitmap fonts. *Machine Translation*, 31(1):35–47.

Ryan Cotterell, Sabrina J. Mielke, Jason Eisner, and Brian Roark. 2018. Are all languages equally hard to language-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)*, pages 536–541, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas M. Cover and Joy A. Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 21–30.

Falcon Dai and Zheng Cai. 2017. Glyph-aware embedding of Chinese characters. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 64–69, Copenhagen, Denmark. Association for Computational Linguistics.

Mathieu Dehouck and Pascal Denis. 2018. A Framework for Understanding the Role of Morphology in Universal Dependency Parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2864–2870, Brussels, Belgium. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh. 2019. A call for prudent choice of subword merge operations in neural machine translation. In *Proceedings of Machine Translation Summit XVII Volume 1: Research Track*, pages 204–213, Dublin, Ireland. European Association for Machine Translation.

Miguel Domingo, Mercedes Garcıa-Martınez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. 2018. How much does tokenization affect neural machine translation? *arXiv preprint arXiv:1812.08621*.

Bonaventure F. P. Dossou and Chris C. Emezue. 2021. Crowdsourced phrase-based tokenization for low-resourced neural machine translation: The case of fon language.

Yerai Doval and Carlos Gómez-Rodríguez. 2019. Comparing neural- and n-gram-based language models for word segmentation. *Journal of the Association for Information Science and Technology*, 70(2):187–197.

C. M. Downey, Fei Xia, Gina-Anne Levow, and Shane Steinert-Threlkeld. 2021. A masked segmental language model for unsupervised natural language segmentation.

Markus Dreyer and Jason Eisner. 2011. Discovering morphological paradigms from plain text using a Dirichlet process mixture model. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 616–627, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Chris Drummond. 2009. Replicability is not reproducibility: Nor is it good science. In *Proceedings of the Evaluation Methods for Machine Learning Workshop at the 26th ICML*.

Matthew S. Dryer, David Gil, Bernard Comrie, Hagen Jung, Claudia Schmidt, et al. 2005. The world atlas of language structures.

Matthew S. Dryer and Martin Haspelmath, editors. 2013. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Hicham El Boukkouri, Olivier Ferret, Thomas Lavergne, Hiroshi Noji, Pierre Zweigenbaum, and Jun'ichi Tsujii. 2020. CharacterBERT: Reconciling ELMo and BERT for word-level open-vocabulary representations from characters. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6903–6915, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Salah El Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 493–499, Cambridge, MA, USA. MIT Press.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14(2):179–211.

Micha Elsner, Sharon Goldwater, Naomi Feldman, and Frank Wood. 2013. A joint learning model of word segmentation, lexical acquisition, and phonetic variability. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 42–54, Seattle, Washington, USA. Association for Computational Linguistics.

Émile Enguehard, Yoav Goldberg, and Tal Linzen. 2017. Exploring the syntactic abilities of RNNs with multi-task learning. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 3–14. Association for Computational Linguistics.

Lawrence Fenton. 1960. The sum of log-normal probability distributions in scatter transmission systems. *IRE Transactions on Communications Systems*, 8(1):57–67.

Richard Futrell, Kyle Mahowald, and Edward Gibson. 2015. Large-scale evidence of dependency length minimization in 37 languages. *Proceedings of the National Academy of Sciences*, 112(33):10336–10341.

Philip Gage. 1994. A new algorithm for data compression. *C Users J.*, 12(2):23–38.

Marco Gaido, Beatrice Savoldi, Luisa Bentivogli, Matteo Negri, and Marco Turchi. 2021. How to split: the effect of word segmentation on gender bias in speech translation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3576–3589, Online. Association for Computational Linguistics.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. 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)*, pages 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Yingqiang Gao, Nikola I Nikolov, Yuhuang Hu, and Richard HR Hahnloser. 2020. Character-level translation with self-attention. *arXiv preprint arXiv:2004.14788*.

S. Geman and D. Geman. 1984. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

S. Goldwater, T. L. Griffiths, and M. Johnson. 2006a. Contextual dependencies in unsupervised word segmentation. In *Proc. of COLING-ACL*.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2009. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2011. Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of Machine Learning Research*, 12(Jul):2335–2382.

Sharon Goldwater, Mark Johnson, and Thomas Griffiths. 2006b. Interpolating between types and tokens by estimating power-law generators. In *Advances in Neural Information Processing Systems*, volume 18. MIT Press.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online. Association for Computational Linguistics.

Johannes Graën, Dolores Batinic, and Martin Volk. 2014. Cleaning the Europarl corpus for linguistic applications. In *Konvens*, pages 222–227.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2016. Improving neural language models with a continuous cache. In *International Conference on Learning Representations 2016*.

Edouard Grave, Sainbayar Sukhbaatar, Piotr Bojanowski, and Armand Joulin. 2019. Training hybrid language models by marginalizing over segmentations. In *Proceedings of the 57th*

Annual Meeting of the Association for Computational Linguistics, pages 1477–1482, Florence, Italy. Association for Computational Linguistics.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Peter J. Green. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732.

Rohit Gupta, Laurent Besacier, Marc Dymetman, and Matthias Gallé. 2019. Character-based nmt with transformer. *arXiv preprint arXiv:1911.04997*.

Ximena Gutierrez-Vasques, Christian Bentz, Olga Sozinova, and Tanja Samardzic. 2021. From characters to words: the turning point of BPE merges. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3454–3468, Online. Association for Computational Linguistics.

W. K. Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic programming encoding for subword segmentation in neural machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

James Henderson. 2020. The unstoppable rise of computational linguistics in deep learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6294–6306, Online. Association for Computational Linguistics.

Tatsuya Hiraoka, Hiroyuki Shindo, and Yuji Matsumoto. 2019. Stochastic tokenization with a language model for neural text classification. In *Proceedings of the 57th Annual*

Meeting of the Association for Computational Linguistics, pages 1620–1629, Florence, Italy. Association for Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2020. Optimizing word segmentation for downstream task. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1341–1351, Online. Association for Computational Linguistics.

Tatsuya Hiraoka, Sho Takase, Kei Uchiumi, Atsushi Keyaki, and Naoaki Okazaki. 2021. Joint optimization of tokenization and downstream model. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 244–255, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Charles F. Hockett. 1960. The origin of speech. *Scientific American*, 203(3):88–97.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Po-Sen Huang, Chong Wang, Sitao Huang, Dengyong Zhou, and Li Deng. 2018. Towards neural phrase-based machine translation. In *International Conference on Learning Representations*.

Kyuyeon Hwang and Wonyong Sung. 2017. Character-level language modeling with hierarchical recurrent neural networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5720–5724. IEEE.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. Automatic Evaluation of Translation Quality for Distant Language Pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952, Cambridge, MA. Association for Computational Linguistics.

Itay Itzhak and Omer Levy. 2021. Models in a spelling bee: Language models implicitly learn the character composition of tokens.

F. Jelinek. 1976. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556.

Miguel Ángel Jiménez-Montaño. 1984. On the syntactic structure of protein sequences and the concept of grammar complexity. *Bulletin of Mathematical Biology*, 46(4):641–659.

Mark Johnson and Sharon Goldwater. 2009. Improving nonparameteric Bayesian inference: experiments on unsupervised word segmentation with adaptor grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 317–325, Boulder, Colorado. Association for Computational Linguistics.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. 2007. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. In *Advances in Neural Information Processing Systems 19*, pages 641–648. MIT Press.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling.

Ákos Kádár, Marc-Alexandre Côté, Grzegorz Chrupała, and Afra Alishahi. 2018. Revisiting the hierarchical multiscale lstm. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3215–3227. Association for Computational Linguistics.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. In *Proceedings of the 55th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 1492–1502, Vancouver, Canada. Association for Computational Linguistics.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2019. Learning to discover, ground and use words with segmental neural language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6429–6441, Florence, Italy. Association for Computational Linguistics.

Eugene Kharitonov, Marco Baroni, and Dieuwke Hupkes. 2021. How bpe affects memorization in transformers.

Aleksandr E. Kibrik. 2017. *Archi (Caucasian - Daghestanian)*, chapter 23. John Wiley & Sons, Ltd.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander Rush. 2016. Character-aware neural language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Diederick P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Christo Kirov, Ryan Cotterell, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Arya McCarthy, Sabrina J. Mielke, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2018. Unimorph 2.0: Universal morphology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC)*. European Language Resources Association (ELRA).

Christo Kirov, John Sylak-Glassman, Rebecca Knowles, Ryan Cotterell, and Matt Post. 2017. A rich morphological tagger for English: Exploring the cross-linguistic tradeoff between morphology and syntax. In *Proceedings of the 15th Conference of the European Chapter*

of the Association for Computational Linguistics: Volume 2, Short Papers, pages 112–117. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the ACL*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, pages 181–184.

Philipp Koehn. 2004. Statistical Significance Tests for Machine Translation Evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, volume 5, pages 79–86.

Philipp Koehn. 2009. *Statistical Machine Translation*. Cambridge University Press.

Philipp Koehn, Alexandra Birch, and Ralf Steinberger. 2009. 462 Machine Translation Systems for Europe. In *proceedings of the twelfth Machine Translation Summit*, pages 65–72.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Segmental recurrent neural networks. In *International Conference on Learning Representations*.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. 2014. A clockwork rnn. In *Proceedings of Machine Learning Research*, volume 32, pages 1863–1871, Bejing, China. PMLR.

Charles J. Kowalski. 1972. On the effects of non-normality on the distribution of the sample product-moment correlation coefficient. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 21(1):1–12.

Julia Kreutzer and Artem Sokolov. 2018. Learning to Segment Inputs for NMT Favors Character-Level Processing. In *Proceedings of the 15th International Workshop on Spoken Language Translation (IWSLT)*.

Taku Kudo. 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)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

D. Robert Ladd. 2012. What is duality of patterning, anyway? *Language and Cognition*, 4(4):261–273.

N Jesper Larsson and Alistair Moffat. 2000. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5(0):365–378.

Gennadi Lembersky, Noam Ordan, and Shuly Wintner. 2012. Adapting translation models to translationese improves SMT. In *Proceedings of EACL*, pages 255–265.

Jindřich Libovický and Alexander Fraser. 2020. Towards reasonably-sized character-level transformer NMT by finetuning subword systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2572–2579, Online. Association for Computational Linguistics.

Jindřich Libovický, Helmut Schmid, and Alexander Fraser. 2021. Why don't people use character-level machine translation?

Yu-Hsiang Lin, Chian-Yu Chen, Jean Lee, Zirui Li, Yuyan Zhang, Mengzhou Xia, Shruti Rijhwani, Junxian He, Zhisong Zhang, Xuezhe Ma, Antonios Anastasopoulos, Patrick Littell, and Graham Neubig. 2019. Choosing Transfer Languages for Cross-Lingual Learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3125–3135, Florence, Italy. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016a. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016b. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association of Computational Linguistics*, 4:521–535.

Patrick Littell, David R. Mortensen, Ke Lin, Katherine Kairis, Carlisle Turner, and Lori Levin. 2017. URIEL and lang2vec: Representing languages as typological, geographical, and phylogenetic vectors. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 8–14, Valencia, Spain. Association for Computational Linguistics.

Frederick Liu, Han Lu, Chieh Lo, and Graham Neubig. 2017a. Learning character-level compositionality with visual features. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2059–2068, Vancouver, Canada. Association for Computational Linguistics.

Haitao Liu. 2008. Dependency distance as a metric of language comprehension difficulty. *Journal of Cognitive Science*, 9(2):159–191.

Haitao Liu, Chunshan Xu, and Junying Liang. 2017b. Dependency distance: A new perspective on syntactic patterns in natural languages. *Physics of Life Reviews*, 21:171–193.

Chi-kiu Lo. 2019. YiSi - a Unified Semantic MT Quality Evaluation and Estimation Metric for Languages with Different Levels of Available Resources. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 507–513, Florence, Italy. Association for Computational Linguistics.

Kaiji Lu, Piotr Mardziel, Fangjing Wu, Preetam Amancharla, and Anupam Datta. 2020. *Gender Bias in Neural Natural Language Processing*. Springer International Publishing, Cham.

Zhaoxin Luo and Michael Zhu. 2021. Recurrent neural networks with mixed hierarchical structures for natural language processing. *CoRR*, abs/2106.02562.

Minh-Thang Luong and Christopher D. Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 1054–1063, Berlin, Germany. Association for Computational Linguistics.

Wentao Ma, Yiming Cui, Chenglei Si, Ting Liu, Shijin Wang, and Guoping Hu. 2020. CharBERT: Character-aware Pre-trained Language Model. In *COLING*.

David J. C. MacKay and Linda C. Bauman Peto. 1995. A hierarchical Dirichlet language model. *Journal of Natural Language Engineering*, 1(3):289–308.

Elman Mansimov, Mitchell Stern, Mia Chen, Orhan Firat, Jakob Uszkoreit, and Puneet Jain. 2020. Towards end-to-end in-image neural machine translation. In *Proceedings of the First International Workshop on Natural Language Processing Beyond Text*, pages 70–74, Online. Association for Computational Linguistics.

Carl de Marcken. 1996. Linguistic structure as composition and perturbation. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

A. A. Markov. 2006. An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. *Science in Context*, 19(4):591–600.

André Martinet. 1949. La double articulation linguistique. *Travaux du Cercle linguistique de Copenhague*, 5:30–37.

Austin Matthews, Graham Neubig, and Chris Dyer. 2018. Using morphological knowledge in open-vocabulary neural language models. In *HLT-NAACL*.

Thomas Mayer and Michael Cysouw. 2014. Creating a massively parallel Bible corpus. In *Proceedings of LREC*, pages 3158–3163.

John McCarthy and Alan Prince. 1999. Faithfulness and identity in prosodic morphology. In R. Kager, H. van der Hulst, and W. Zonneveld, editors, *The Prosodic Morphology Interface*, pages 269–309. Cambridge University Press.

John McWhorter. 2001. The world's simplest grammars are creole grammars. *Linguistic Typology*, 5(2):125–66.

Yuxian Meng, Wei Wu, Fei Wang, Xiaoya Li, Ping Nie, Fan Yin, Muyu Li, Qinghong Han, Xiaofei Sun, and Jiwei Li. 2019. Glyce: Glyph-vectors for chinese character representations. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017a. Regularizing and optimizing LSTM language models. *arXiv preprints*, abs/1708.02182.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017b. Pointer sentinel mixture models. In *Proc. International Conference on Learning Representations*, Toulon, France.

Bart van Merriënboer, Amartya Sanyal, Hugo Larochelle, and Yoshua Bengio. 2017. Multiscale sequence modeling with a learned dictionary. In *MLSLP*.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

Sabrina J. Mielke. 2019. Can you compare perplexity across different segmentations?

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. *CoRR*, abs/2112.10508.

Sabrina J. Mielke, Ryan Cotterell, Kyle Gorman, Brian Roark, and Jason Eisner. 2019. What kind of language is hard to language-model? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4975–4989, Florence, Italy. Association for Computational Linguistics.

Sabrina J. Mielke and Jason Eisner. 2019. Spell once, summon anywhere: A two-level open-vocabulary language model. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6843–6850.

Sabrina J. Mielke, Arthur Szlam, Emily Dinan, and Y-Lan Boureau. 2022. Reducing Conversational Agents' Overconfidence Through Linguistic Calibration. *Transactions of the Association for Computational Linguistics*, 10:857–872.

Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association*, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 3111–3119.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan

Černocký. 2012. Subword language modeling with neural networks. *preprint (rejected from ICASSP 2012)*.

Daichi Mochihashi, Takeshi Yamada, and Naonori Ueda. 2009. Bayesian unsupervised word segmentation with nested pitman-yor language modeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 100–108, Suntec, Singapore. Association for Computational Linguistics.

Amir More, Özlem Çetinoğlu, Çağrı Çöltekin, Nizar Habash, Benoît Sagot, Djamé Seddah, Dima Taji, and Reut Tsarfaty. 2018. CoNLL-UL: Universal morphological lattices for Universal Dependency parsing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Thomas Müller, Hinrich Schütze, and Helmut Schmid. 2012. A comparative investigation of morphological language modeling for the languages of the European Union. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 386–395, Montréal, Canada. Association for Computational Linguistics.

NIST Multimodal Information Group. 2010a. NIST 2002 Open Machine Translation (OpenMT) evaluation LDC2010T10.

NIST Multimodal Information Group. 2010b. NIST 2003 Open Machine Translation (OpenMT) evaluation LDC2010T11.

NIST Multimodal Information Group. 2010c. NIST 2004 Open Machine Translation (OpenMT) evaluation LDC2010T12.

NIST Multimodal Information Group. 2010d. NIST 2005 Open Machine Translation (OpenMT) evaluation LDC2010T14.

NIST Multimodal Information Group. 2010e. NIST 2006 Open Machine Translation (OpenMT) evaluation LDC2010T17.

NIST Multimodal Information Group. 2010f. NIST 2008 Open Machine Translation (OpenMT) evaluation LDC2010T21.

NIST Multimodal Information Group. 2010g. NIST 2009 Open Machine Translation (OpenMT) evaluation LDC2010T23.

NIST Multimodal Information Group. 2013a. NIST 2008-2012 Open Machine Translation (OpenMT) progress test sets LDC2013T07.

NIST Multimodal Information Group. 2013b. NIST 2012 Open Machine Translation (OpenMT) evaluation LDC2013T03.

Vít Novotný, Eniafe Festus Ayetiran, Dávid Lupták, Michal Stefánik, and Petr Sojka. 2021. One size does not fit all: Finding the optimal n-gram sizes for fasttext models across languages. *CoRR*, abs/2102.02585.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, An-

dreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Lewis M. Paul, Gary F. Simons, Charles D. Fennig, et al. 2009. *Ethnologue: Languages of the world*, 19 edition. SIL International, Dallas.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. 2017. Mimicking word embeddings using subword RNNs. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 102–112, Copenhagen, Denmark. Association for Computational Linguistics.

Jim Pitman and Marc Yor. 1997. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 25(2):855 – 900.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany. Association for Computational Linguistics.

Matt Post. 2018. A Call for Clarity in Reporting BLEU Scores. In *Proceedings of the Third*

Conference on Machine Translation: Research Papers, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Ella Rabinovich and Shuly Wintner. 2015. Unsupervised identification of translationese. *Transactions of the Association for Computational Linguistics*, 3:419–432.

Ella Rabinovich, Shuly Wintner, and Ofek Luis Lewinsohn. 2016. A parallel corpus of translationese. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 140–155. Springer.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *preprint*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Shauli Ravfogel, Yoav Goldberg, and Tal Linzen. 2019. Studying the inductive biases of RNNs with synthetic variations of natural languages. 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)*, pages 3532–3542, Minneapolis, Minnesota. Association for Computational Linguistics.

Philip Resnik, Mari Broman Olsen, and Mona Diab. 1999. The bible as a parallel corpus: Annotating the 'book of 2000 tongues'. *Computers and the Humanities*, 33(1):129–153.

Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407.

José Carlos Rosales Núñez, Guillaume Wisniewski, and Djamé Seddah. 2021. Noisy UGC translation at the character level: Revisiting open-vocabulary capabilities and robustness of char-based models. In *Proceedings of the Seventh Workshop on Noisy User-generated Text (W-NUT 2021)*, pages 199–211, Online. Association for Computational Linguistics.

Benoît Sagot. 2013. Comparing complexity measures. In *Computational Approaches to Morphological Complexity*.

Benoît Sagot and Pierre Boullier. 2008. SxPipe 2: architecture pour le traitement pré-syntaxique de corpus bruts. *Revue TAL*, 49(2):155–188.

Elizabeth Salesky, David Etter, and Matt Post. 2021. Robust open-vocabulary translation from visual text representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7235–7252, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Elizabeth Salesky, Andrew Runge, Alex Coda, Jan Niehues, and Graham Neubig. 2018. Optimizing segmentation granularity for neural machine translation. *arXiv preprint arXiv:1810.08641*.

Cicero dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826.

Ferdinand de Saussure. 1916. *Course in General Linguistics*. Columbia University Press. English edition of June 2011, based on the 1959 translation by Wade Baskin.

Jürgen Schmidhuber. 1991. Neural sequence chunkers.

Jürgen Schmidhuber. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

S. C. Schwartz and Y. S. Yeh. 1982. On the distribution function and moments of power sums with log-normal components. *The Bell System Technical Journal*, 61(7):1441–1462.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

Claude E. Shannon. 1951. Prediction and entropy of printed English. *Bell Labs Technical Journal*, 30(1):50–64.

Pamela Shapiro and Kevin Duh. 2018. BPE and charcnns for translation of morphology: A cross-lingual comparison and analysis. *CoRR*, abs/1809.01301.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A Study of Translation Edit Rate with Targeted Human Annotation. In *Proceedings of Association for Machine Translation in the Americas*.

Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. 2021. Fast WordPiece tokenization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2089–2103, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Richard Sproat, Bruno Cartoni, HyunJeong Choe, David Huynh, Linne Ha, Ravindran

Rajakumar, and Evelyn Wenzel-Grondie. 2014. A database for measuring linguistic information content. In *LREC*.

James A. Storer and Thomas G. Szymanski. 1982. Data compression via textual substitution. *J. ACM*, 29(4):928–951.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: Trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of LREC*, pages 4290–4297.

Milan Straka and Jana Straková. 2017. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *CoNLL 2017 Shared Task: Multilingual parsing from raw text to Universal Dependencies*, pages 88–99.

Sara Stymne. 2017. The Effect of Translationese on Tuning for Statistical Machine Translation. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 241–246, Gothenburg, Sweden. Association for Computational Linguistics.

Baohua Sun, Lin Yang, Catherine Chi, Wenhan Zhang, and Michael Lin. 2019. Squared english word: A method of generating glyph to use super characters for sentiment analysis. In *Proceedings of the 2nd Workshop on Affective Content Analysis (AffCon 2019) co-located with the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019)*, volume 2328, pages 140–151.

Baohua Sun, Lin Yang, Patrick Dong, Wenhan Zhang, Jason Dong, and Charles Young. 2018. Super characters: A conversion from sentiment classification to image classification. In *Proceedings of the 9th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 309–315, Brussels, Belgium. Association for Computational Linguistics.

Zhiqing Sun and Zhi-Hong Deng. 2018. Unsupervised neural word segmentation for Chinese via segmental language modeling. In *Proceedings of the 2018 Conference on*

Empirical Methods in Natural Language Processing, pages 4915–4920, Brussels, Belgium. Association for Computational Linguistics.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1017–1024, New York, NY, USA. ACM.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Dan Tito Svenstrup, Jonas Hansen, and Ole Winther. 2017. Hash embeddings for efficient word representations. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2021. Charformer: Fast character transformers via gradient-based subword tokenization.

Yee Whye Teh. 2006. A hierarchical bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 985–992, Sydney, Australia. Association for Computational Linguistics.

Antonio Toral and Víctor M. Sánchez-Cartagena. 2017. A Multifaceted Evaluation of Neural versus Phrase-Based Machine Translation for 9 language Directions. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1063–1073, Valencia, Spain. Association for Computational Linguistics.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden. Association for Computational Linguistics.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for Neural Machine Translation. *CoRR*, abs/1803.07416.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

David Vilar and Marcello Federico. 2021. A statistical extension of byte-pair encoding. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 263–275, Bangkok, Thailand (online). Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2019b. Neural machine translation with byte-level subwords.

Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng. 2017. Sequence modeling via segmentations. In *International Conference on Machine Learning*, pages 3674–3683.

Dingquan Wang and Jason Eisner. 2016. The Galactic Dependencies Treebanks: Getting More Data by Synthesizing New Languages. *Transactions of the Association for Computational Linguistics*, 4:491–505.

Haohan Wang, Peiyan Zhang, and Eric P Xing. 2020. Word shape matters: Robust machine translation with visual embedding. *arXiv preprint arXiv:2010.09997*.

Lihao Wang, Zongyi Li, and Xiaoqing Zheng. 2021. Unsupervised word segmentation with bi-directional neural language model. *CoRR*, abs/2103.01421.

Jonathan J. Webster and Chunyu Kit. 1992. Tokenization as the initial phase in NLP. In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*.

Greg C. G. Wei and Martin A. Tanner. 1990. A monte carlo implementation of the em algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704.

Max Welling and Yee Whye Teh. 2011. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA. Omnipress.

Lawrence Wolf-Sonkin, Jason Naradowsky, Sabrina J. Mielke, and Ryan Cotterell. 2018. A structured variational autoencoder for contextual morphological inflection. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2631–2641, Melbourne, Australia. Association for Computational Linguistics.

J Gerard Wolff. 1975. An algorithm for the segmentation of an artificial language analogue. *British Journal of PsychologyJ*, 66:79–90.

F. Wood, J. Gasthaus, C. Archambeau, L. James, and Y.W. Teh. 2011. The sequence memoizer. *Communications of the ACM*, 54(2):91–98.

Frank Wood and Yee Whye Teh. 2008. A hierarchical, hierarchical pitman yor process language model. In *Proceedings of the ICML/UAI Workshop on Nonparametric Bayes*. Citeseer.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*.

Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 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)*, pages 7361–7373, Online. Association for Computational Linguistics.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2021. Byt5: Towards a token-free future with pre-trained byte-to-byte models.

David Yarowsky, Grace Ngai, and Richard Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the First International Conference on Human Language Technology Research*, pages 1–8.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.

Mike Zhang and Antonio Toral. 2019. The Effect of Translationese in Machine Translation Test Sets. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 73–81, Florence, Italy. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Xinsong Zhang, Pengshuai Li, and Hang Li. 2021. AMBERT: A pre-trained language model with multi-grained tokenization. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 421–435, Online. Association for Computational Linguistics.

J. Ziv and A. Lempel. 1978. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536.

Ran Zmigrod, Sabrina J. Mielke, Hanna Wallach, and Ryan Cotterell. 2019. Counterfactual data augmentation for mitigating gender stereotypes in languages with rich morphology. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1651–1661, Florence, Italy. Association for Computational Linguistics.