MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Morphology in the Age of Pre-trained Language Models

Ph.D. Dissertation

Judit Ács

Advisor:
András Kornai D.Sc.

Budapest
2025

Judit Ács
January 2025

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

H-1117 Budapest, Magyar tudósok körútja 2. (Bldg. Q.)

Declaration of own work and references

I, Judit Ács, hereby declare that this dissertation, and all results claimed therein are my own work, and rely solely on the references given. All segments taken word-by-word, or in the same meaning from others have been clearly marked as citations and included in the references.

Nyilatkozat önálló munkáról, hivatkozások átvételéről

Alulírott Ács Judit kijelentem, hogy ezt a doktori értekezést magam készítettem és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2025. 01. 20.

Ács Judit

Acknowledgements

First and foremost I thank my advisor, András Kornai for his continuous support throughout my years as a PhD student. I would not have started this journey without the opportunity to join his team in 2012 at SZTAKI. He introduced me to the world of NLP and his research attitude and integrity helped me form my identity as a researcher and a professional.

I want to thank the members of my first research group at SZTAKI including Gábor Recski and Attila Zséder, who were my first guides in my professional career. I thank the rest of the team, Katalin Pajkossy, Márton Makrai, Dávid Nemeskey and Eszter Simon for their professional and personal support.

I am eternally grateful to Noah Smith who accepted me as a visiting student through the Fulbright Scholarship. My year in his group is an experience I will cherish for my life.

I want to thank my numerous coauthors and friends. The list is too long to name everyone but I remember every piece of encouragement and I am glad that we had a chance to work together.

Finally I want to thank my family. My husband, Gábor, who put up with my late hours, my hectic schedule and my professional and personal struggles. I also thank him for joining me on my Fulbright program in Seattle. I thank my son, Nándor and his soon-to-be-born little brother who continue to be the light of my day. I want to thank my parents, siblings, grandparents and other family members who were always proud of me despite my inability to explain my research to them. I want to thank my mother for her support and I am devastated that she is no longer here to witness my graduation. I dedicate this work to her.

Summary

The field of natural language processing (NLP) has adopted deep learning methods in the past 15 years. Nowadays the state-of-the-art in most NLP tasks is some kind of neural model, often the fine-tuned version of a pre-trained language model. The efficacy of these models is demonstrated on various English benchmarks and increasingly, other monolingual and multimultilingual benchmarks. In this dissertation I explore the application of deep learning models on low level tasks, particularly morphosyntactic tasks in multiple languages.

The first part of this dissertation (Chapters 3 and 4) explores the application of deep learning models for classical morphosyntactic tasks such as morphological analysis and generation in dozens of languages with special focus on Hungarian.

The second part of this dissertation (Chapters 5 to 8) deals with pre-trained language models, mostly models from the BERT family. I include some experiments on GPT-4o and GPT-4o-mini. These models show excellent performance on various tasks in English and some high density languages. However, their evaluation in medium and low density languages is lacking. I present a methodology for generating morphosyntactic benchmarks in arbitrary languages and I analyze multiple BERT-like models in detail. My main tool for analysis is the probing methodology which I extend the with perturbations, the systematic removal of certain information from the sentence. I use Shapley values to further refine my analysis.

Összefoglaló

A természetes nyelvfeldolgozás (NLP) területe az elmúlt 15 évben átvette a mélytanulási módszereket. Manapság a legtöbb NLP feladatban valamilyen neurális modell adja a legjobb megoldást, gyakran egy előre betanított nyelvi modell finomhangolt változata. E modellek hatékonyságát különféle angol nyelvű benchmarkokon, valamint egyre inkább más egynyelvű és többnyelvű benchmarkokon is igazolják. Ebben a disszertációban a mélytanulási modellek alkalmazását vizsgálom alacsony szintű, elsősorban morfoszintaktikai feladatokra több nyelven.

A disszertáció első része (3. és 4 fejezet) a mélytanulási modellek alkalmazását tárgyalja klasszikus morfoszintaktikai feladatokhoz, például morfológiai elemzéshez és inflexióhoz több tucat nyelven, különös tekintettel a magyar nyelvre.

A disszertáció második része (5–8. fejezet) az előre betanított nyelvi modellekkel foglalkozik, elsősorban a BERT-család modelljeivel. Néhány kísérletet bemutatok a GPT-4o és a GPT-4o-mini modellekkel is. Ezek a modellek kiváló teljesítményt nyújtanak különböző angol és más sok erőforrással ellátott feladatokban. Ugyanakkor a közepes és kevés erőforrással rendelkező nyelveken történő értékelésük hiányos. Bemutatok egy módszertant a morfoszintaktikai benchmarkok generálására tetszőleges nyelven, valamint részletesen elemzek több BERT modellt. Elemzéseim fő eszköze a szondázó (probing) módszertan, melyet kiterjesztek pertubációkkal, vagyis bizonyos információk szisztematikus eltávolításával a mondatokból. Az elemzésem további finomításához Shapley-értékeket használok.

Contents

Chapter 1

Introduction

The study of morphology, the internal structure of words, dates back to ancient times. The Sanskrit grammarian, Pāṇini formulated 3,959 rules for Sanskrit morphology sometime between the 7th and 4th century BCE. The Greco-Roman grammatical tradition included morphological analysis, and Arabic morphology dates back to at least 1200 CE. These early foundational works continue to influence linguistics today.

The rise of computational linguistics has had a profound impact on morphology. With the advent of natural language processing (NLP), researchers have developed algorithms to analyze and generate morphological structures in diverse languages. These technologies underpin applications such as machine translation, speech recognition, and text analysis.

However, the introduction of deep neural networks and the end-to-end training of models led to a decline in the need for separate morphological analysis. This is evident from the disappearance of morphology-related workshops and the ever decreasing number of papers about computational morphology at major conferences. The main question of this thesis is whether we still need to concern ourselves about morphology or is the knowledge simply learned by language models trained on sufficiently large raw data. We try to answer this question in two parts. First, we examine small neural networks on specific morphological tasks, then we turn to pre-trained language models and assess how well they acquire morphology during training. Remarkably, the end-to-end trained models perform much better than the traditional morphology modules, which suggests that not just a separate morphology, but the entire 'modular' view may be a thing of the past.

This thesis is organized as follows. Chapter 2 gives a general background for the deep learning models we use throughout the thesis (Section 2.2). The thesis's main research area is computational morphology, so we provide a short introduction to morphology and morphosyntax here as well (Section 2.3).

In Chapter 3 we demonstrate the effectiveness of attention-based models on morphosyntactic tasks first in Hungarian, then in over 100 languages as a submission to the SIGMORPHON18 Shared Tasks.

In Chapter 4 we explore SoPa, a differentiable restricted finite state automaton capable of learning which spans of the input is important to the output. We apply this model to morphology and compare the learned spans to morphemes.

Chapter 5 gives an introduction to the second part of this thesis which deals with pre-trained language models (PLMs) including some large language models (LLMs).

Chapter 6 describes our extensive probing experiments. We first introduce morphosyntactic probing as an evaluation method (Section 6.2). We created a large-scale multilingual dataset (Section 6.3) which we applied for multiple pre-trained models. We perform in-depth analysis on mBERT and XLM-RoBERTa, two massively multilingual models (Section 6.5). These models use subword tokenizers which require a way of pooling multiple subword representations that correspond to a single token. We explore these options in Section 6.4. The probing methodology has its fair share of criticism which we address with various ablation methods (Section 6.6).

The previous chapters focused on multilingual models. Chapter 7 dives into language specific models, particularly models for Hungarian (Section 7.1) and other Uralic languages (Section 7.2).

Finally, in Chapter 8, we further refine our analysis with perturbations, methods for the systematic removal of some information from the probing data (Section 8.1). We use the results as inputs to clustering algorithms over languages and we show that related languages tend to form clusters (Section 8.1.2). We also apply a game theoretic approach, Shapley values, which allows for fine-grained analysis of morphosyntactic phenomena, while retaining the scope of this multilingual work (Section 8.2).

The dissertation's main contributions are summarized in Theses 1 to 5.

Thesis 1

I demonstrated that encoder-decoder (a.k.a. sequence-to-sequence or seq2seq) models are well-suited for morphological inflection and generation. This holds for type-level and sentence-level tasks in multiple languages.

Subtheses:

1.1 I collected and prepared a silver standard Hungarian dataset for morphological inflection and analysis using a high quality rule-based analyzer.

1.2 I implemented two types of sequence-to-sequence or encoder-decoder models with attention: LSTM with soft attention and LSTM with hard attention.

1.3 I trained and evaluated the models on the Hungarian dataset.

1.4 I developed two new types of encoder-decoder models for morphological inflection. The first model is a double encoder-single decoder model for type-level inflection. The second model is complex multi-encoder model for sentence-level inflection. I participated in the CoNLL-SIGMORPHON 2018 Shared Task with these models as an individual team. Task 1 was type-level inflection in over 100 languages, Task 2 was inflection in context (sentence) in 7 languages. I placed 3rd and 2nd in the two tasks respectively with multi-encoder and single-decoder seq2seq neural networks.

These contributions were published in Ács (2018) and they are my sole contribution (Chapter 3).

Thesis 2

I adapted a differentiable weighted finite state RNN to sequence-to-sequence tasks and showed that neural pattern matching can extract morphosyntactic patterns in multiple languages when used as an encoder for morphological inflection and analysis.

Subtheses:

2.1 I reimplemented the Soft Patterns or SoPa neural model (Schwartz et al., 2018), a restricted and differentiable finite state automaton model originally used for text classification tasks. I added an LSTM decoder and used SoPa as an encoder-decoder model. This model uses pattern matching on the encoder side.

2.2 I applied the model to type-level morphological analysis and inflection in 12 typologically diverse languages.

2.3 I extracted patterns (character sequences) from trained SoPa models and manually examined their linguistic plausibility.

2.4 I introduced a model similarity metric defined for a pair of SoPa models. This metric allows comparing different tasks that use the same input data. The higher the similarity between two tasks, the more likely they are to rely on the same patterns.

These contributions were published in Ács and Kornai (2020). The paper was awarded the best paper award at the Hungarian Computational Linguistics Conference in 2020. The implementation is entirely my contribution (Chapter 4).

Thesis 3

I developed a new methodology for morphosyntactic probing. It relies on CoNLL-U formatted data which is widely available in the Universal Dependencies Treebanks, therefore my methodology is applicable to a large number of languages and morphosyntactic tags. Using my probing methodology I showed that pre-trained language models (PLMs) trained on unannotated text learn morphology. PLMs' representations retain morphosyntactic information across a large set of typologically diverse languages and multiple tasks.

Subtheses:

3.1 I introduced the largest multilingual morphosyntactic probing dataset with 247 tasks in 42 families from 10 language families. I define a probing sample as a triplet of a sentence, a particular token called *target token* in the sentence and a morphosyntactic tag corresponding to the target token. I used the Universal Dependencies Treebank (Nivre et al., 2018) to generate probing samples according to this definition.

3.2 I evaluated 6 multilingual PLMs and analyze two, mBERT and XLM-RoBERTa in detail. I compared them to various baselines and show that PLMs indeed learn morphosyntactic information.

3.3 I evaluated GPT-4o and GPT-4o-mini on the test set of the probing dataset via prompting.

3.4 I examined the tokenizer of PLMs and showed that although multilingual models support over 100 languages, the tokenizer works better on languages that use the Latin script, particularly English. (Ács, 2019)

3.5 Probing as an analysis tool for blackbox models has been criticized for various reasons (Belinkov, 2021). I introduced more than 10 ablation methods and showed that the conclusions drawn from morphosyntactic probing are robust.

3.6 The token-level usage of PLMs requires a way of handling tokens split into multiple subwords represented by multiple vectors. A pooling function (parametric on non-parametric) may be used to infer a single vectors by token. I showed that the choice of pooling function matters, especially for feature extraction (when the PLM is not fine-tuned). I compared 9 pooling functions in 7 languages and 3 tasks (Ács et al., 2021a).

These contributions were published in (Ács, 2019; Ács et al., 2021a; Ács et al., 2023). The experiment design was done in collaboration with my coauthors and the implementation is my sole contribution (Chapter 6).

Thesis 4

I used my morphosynactic probing dataset and methodology to demonstrate that monolingual PLMs are better in their respective languages than multilingual PLMs but the difference is small and often not statistically significant. Moreover both monolingual and multilingual PLMs can be successfully transferred to new languages as long as the new language uses the same writing system.

Subtheses:

4.1 I analyzed 5 PLMs with Hungarian support and I found that HuBERT (Nemeskey, 2021), a Hungarian-only model is better at morphological, POS and NER tagging than the multilingual models, especially distilBERT, but the margin is small. (Ács et al., 2021)

4.2 I extended this study to the languages of the Uralic family (Ács et al., 2021b). I drew similar conclusions in Estonian and Finnish, the only Uralic languages with dedicated monolingual PLMs, as in Hungarian.

4.3 I trained POS and NER tagging models in minority Uralic languages by transferring multilingual and monolingual models. The cross-language models are surprisingly successful despite the extremely small training data available in some languages. The new models appear to be state-of-the-art without any language-specific effort.

These contributions were published in (Ács et al., 2021; Ács et al., 2021b). The experiment design and the result analysis was done in collaboration with my coauthors. The implementation is my contribution (Chapter 7).

Thesis 5

I refined my probing analysis with perturbations that aim to find the exact location of morphosyntactic information in a sentence. The systematic removal of certain information (perturbations) reveals where the information is stored. I used Shapley values to quantify the role of context in morphosyntax and the results often agree with linguistic intuitions.

Subtheses:

5.1 I introduced a set of perturbation methods that remove some source of information from a sentence. I retrained the morphosyntactic probes (cf. Thesis 3) on the 247 probing tasks. The results offer insight on where the information is stored in the sentence. We can often find linguistic explanations for them.

5.2 I used the results of perturbations as features for clustering the languages. Such clustering tends to group languages from the same family in the same cluster with some notable exceptions. It also tends to cluster typologically similar but unrelated languages in the same cluster.

5.3 I defined a sentence as a 9-player coalition game where the players are tokens or groups of tokens relative to the target token in the morphosyntactic probe. I applied the Shapley framework on each probing task.

5.4 I analyzed the Shapley values from mBERT, XLM-RoBERTa and an LSTM baseline and found that the inter-model correlation is high which demonstrates that the Shapley values are more descriptive of linguistic structure than of the models. I identified the outlier tasks and I found that the Shapley values often confirm the linguistic properties of the particular language and task.

These contributions were published in (Ács et al., 2023). The experiment design and the result analysis was done in collaboration with my coauthors. The implementation is my contribution (Chapter 8).

Chapter 2

Background

This chapter provides an overview of the three principal areas relevant to this thesis. First, we introduce the field of Natural Language Processing (see Section 2.1). Next, we present a concise introduction to the evolving domain of deep learning in Section 2.2. Given that Part I addresses models for low-level morphological tasks without the use of pre-trained models, we defer the discussion of language models to Part II. Finally, Section 2.3 offers a brief overview of natural language morphology, with particular emphasis on Hungarian morphology, as Hungarian serves as a key focus of this research.

2.1 Natural language processing

Natural Language Processing (NLP) is a field of artificial intelligence that enables computers to understand, interpret, and generate human language. It combines concepts from computer science, linguistics, mathematics, and machine learning to create systems that can interact with natural language. Its goal is to bridge the gap between human communication and machine understanding, allowing computers to process large volumes of text data and extract valuable insights.

A foundational challenge in NLP is dealing with the complexity and variability of human language. Language is inherently ambiguous, with words and sentences often carrying multiple meanings depending on context. For instance, the word *bank* can refer to a financial institution or the side of a river, and determining the intended meaning requires context.

NLP research dates back to the 1950s when machine translation was highly desired by political actors. The 1954 Georgetown-IBM experiment demonstrated the automatic translation of 60 Russian sentences to English and fueling further funding for the project but it unfortunately did not live up to the expectations. The development of NLP continued with similar rule-based systems but real breakthroughs waited until the spread of statistical methods in the 1990s. Statistical methods require training corpora which became available with the spread of the world wide web. Machine learning methods such as Naive Bayes, Support Vector Machines, Logistic Regression and Decision Trees showed promising results in both research and industrial applications.

The 2010s welcomed the next breakthrough with the development of word embeddings and the efficient usage of neural networks. We give an overview of neural networks in Section 2.2 and introduce embeddings in more detail in Section 5.1.1.

NLP research defines many tasks at different levels of application. High-level tasks are the ones that the end user is familiar with. Mid and low-level tasks are usually building blocks of complex systems implementing high-level tasks. In the rest of this section we give an overview of the most common tasks in NLP with special emphasis on the ones related to this thesis.

2.1.1 High-level NLP applications

Natural language generation. NLG is an umbrella term for any task where the output is free-form natural language. This output is generally conditioned on the input which can be another piece of natural text, a database, or their combination.

Summarization. Automatic summarization produces a summary of a chunk of text, typically a news article. There are two types of summarization: *extractive summarization* highlights the important parts of the input text, while *abstractive summarization* generates a new summary. Abstractive summarization is a prominent example of NLG.

Machine translation. Machine translation is the automatic translation from one human language to another. It is one of the oldest NLP tasks dating back to the 1950s. Early rule-based methods were first replaced with statistical machine translation and later with neural machine translation.

Question answering. QA systems answer questions posed in natural language. Similarly to summarization we distinguish between extractive and abstractive QA. Another categorization is *open book* vs. *closed book* QA depending on the model's ability to use external data sources (open book).

Sentiment analysis. Sentiment analysis classifies the emotional intent of a text making it an example of text classification in general. The simplest case is the binary classification of *positive* vs. *negative* (or ternary with the inclusion of *neutral*) of single sentences.

2.1.2 Sentence level NLP tasks

We loosely equate mid-level tasks with sentence-level tasks. Mid-level tasks are usually components of high-level applications or used as evaluations of novel models. Along with low-level tasks, these correspond to the traditional fields of linguistics and they are often parts of NLP preprocessing pipelines. Here we describe some of the most commonly used mid-level tasks.

Tokenization. Tokenization breaks down text into smaller parts. It is an important step of most if not all NLP pipelines. Tokenization can be word-level, morpheme-level, subword or character tokenization.

Part-of-speech tagging. POS tagging marks every word with a *part of speech* or *grammatical category*. POS generally describes a word's syntactic function and in some languages, morphological behavior. Some of the most common POS categories are noun, verb, adjective, adverb, and pronoun. Many words, particularly in English, are ambiguous without context (e.g. *divorce* can be a noun or a verb). POS tagging is generally done at the sentence level as the sentential context is enough for disambiguating most words.

Named entity recognition. NER or NER tagging labels words or multiword expressions that refer to proper names. Additionally NER models categorize proper names as persons, locations, organizations, and more.

Word sense disambiguation. Many words have more than one related or unrelated meaning and WSD selects one of them based on context.

Parsing. Parsings maps a sentence to a tree-like structure based on its grammatical structure and the relationship of the tokens. Two main types of parsing are used: *dependency parsing* and *constituency parsing*. Dependency parsing focuses on the relationship of the words to one another, while constituency parsing builds a parse tree that may involve abstract parts of the sentence such as VP (verb phrase).

Mid-level NLP tasks used to be important components of high-level systems, but with the advent of efficient representation learning with neural networks, many of them play a smaller role nowadays. We use POS and NER tagging as evaluation tasks in Chapter 6 and Chapter 7 extensively.

2.1.3 Word level NLP tasks

Word level or low-level tasks are performed on single word forms rather than full sentences or paragraphs.

Lemmatization. Lemmatizers remove inflection (cf. Section 2.3) from words and return the base dictionary form or *lemma*. Their main purpose is to reduce the number of different word forms particularly in agglutinative languages. *Stemming* is a similar process but it relies on simple rules rather than a lexicon.

Morphological segmentation. Morphological segmentation is the process of identifying morpheme boundaries within a word.

Morphological analysis. Morphological analysis describes the internal structure of a word including derivational and inflectional suffixes. Analyzers usually identify a lemma as well. An analysis may or may not contain the morpheme boundaries (segmentation).

Morphological reinflection. Reinflection is the opposite task of morphological analysis. Given a lemma and an inflectional paradigm, the desired output is the inflected form. This task used to play an important role in cross-lingual applications.

This thesis focuses on morphology and we dive into morphological reinflection and analysis, particularly Part I.

2.2 Neural networks

Neural networks date back to the early 20th century with theories on how the brain might work. In 1943 Warren McCulloch and Walter Pitts proposed the *artificial neuron model* inspired by neuropsychology (McCulloch and Pitts, 1943). Based on their work, Rosenblatt (1957) implemented as the *perceptron* algorithm for supervised binary classification. Rosenblatt built the first hardware implementation of the perceptron which became the first single layer neural network formulated as:

$$f(x) = h(w \cdot x + b), \tag{2.1}$$

where x is the input, w is the weight, both are real-valued vectors, $w \cdot x$ is their dot product and b is a scalar bias. h is the unit step function. f is a learned threshold function and its output for x is a single binary value.

Unfortunately the original perceptron could only solve linearly separable problems. This issue was highlighted in the influential 1969 book *Perceptrons* by Marvin Minsky (Minsky and Papert, 1969), particularly perceptron's inability to solve the XOR problem, which led to the so-called *AI Winter*, a widespread withdrawal of research funds in the field.

The field was revitalized in the 1980s with the development of the *backpropagation* algorithm, a training algorithm that allowed multilayer neural networks or *deep neural networks* to learn complex patterns with the introduction of non-linear activation functions.

In the 2000s and 2010s, the combination of advances in hardware, GPUs in particular, the availability of large datasets and new techniques like dropout and rectified linear units (ReLU) fueled the rise of deep learning. Pioneers such as Geoffrey Hinton, Yann LeCun and Yoshua Bengio helped bring neural networks to the forefront of AI research, leading to breakthroughs in image recognition, natural language processing and game-playing AI (reinforcement learning).

2.2.1 Feed-forward neural networks

A *Feedforward Neural Network (FNN)* is the simplest type of artificial neural network where information flows in one direction. There are no cycles or loops, meaning data moves forward only, without feedback. This type of neural network is called *multilayer perceptron* or *MLP* and it is illustrated in Figure 2.1.

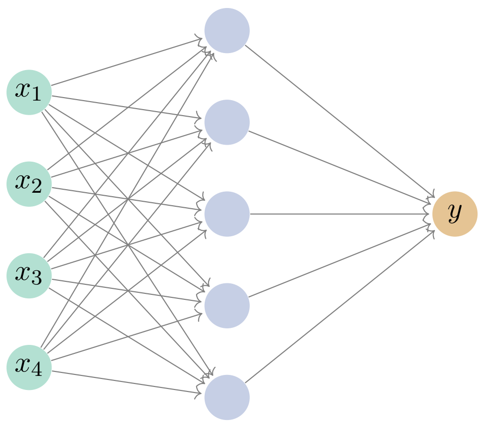

Figure 2.1: Multi-layer perceptron with four inputs in green, a hidden layer with five neurons in blue and one output in brown.

An MLP consists of three types of layers: (i) and input layer which receives the raw data or features, (ii) one or more hidden layers and (iii) an output layer which produces the final result or prediction. In this example in Figure 2.1 there is a single neuron in the output layer, typical of binary classification problems. For multiclass problems MLP's generally have as many output neurons as the number of classes.

Each neuron in a layer is connected to neurons in the next layer, and each connection has a weight, which is adjusted during training. Neurons apply *activation functions* to the weighted sum of their inputs to introduce non-linearity, allowing the network to model more complex patterns. Common activation functions include the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.2}$$

which squashes output between 0 and 1, frequently used in the past but prone to issues like vanishing gradients. A similar squashing function with outputs between -1 and 1 is the hyperbolic tangent:

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3}$$

ReLU or *Rectified Linear Unit* is a piecewise linear activation function defined as:

$$\text{ReLU}(x) = \max(0, x). \tag{2.4}$$

ReLU outputs the input directly if positive, and 0 otherwise. Although ReLU is non-differentiable at 0, it is differentiable everywhere else and its derivative at 0 can be chosen as 0 or 1. ReLU has other variants that solve this problem such as Leaky ReLU.

Activation functions are usually applied after each layer with the exception of the output layer. An MLP with one hidden layer and ReLU activation implements the following function:

$$\text{MLP}(x) = W_2(\text{ReLU}(W_1 x + b_1)) + b_2, \tag{2.5}$$

where the weight matrices W_1 and W_2 and b_1 and b_2 biases are the parameters of the MLP and they are adjusted during training to minimize the error between the predicted and the desired target values. A single layer of an MLP, also called *dense layer*, is a frequent building block of more complex neural networks.

2.2.2 Training neural networks

To train neural networks, we need a *training dataset*, a set of input-output pairs. A smaller part of the training dataset is usually designated as a *validation dataset*, not used for actual weight updates but rather for setting hyperparameters or controlling the training process. Additionally an untouched *test set* is used for the final evaluation of a model. For classification problems the outputs are class labels. Neural networks are universal function approximators (Hornik et al., 1989; Cybenko, 1989; Csáji, 2001) meaning that in theory they can learn any function. Training a neural network involves adjusting its weights and biases to minimize the error between the predicted output and the actual target. The goal is to optimize the network so that it can generalize well to unseen data. The process typically involves three steps: forward pass, loss calculation and backward pass.

During the forward pass the input data is passed through the network layer by layer. Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to the next layer. The output layer produces the network's prediction.

The difference between the predicted output and the actual target is measured using a loss function such as the cross-entropy loss for classification problems, which is defined as:

$$l = \log \frac{\exp y_k}{\sum_{c=1}^{C} \exp y_c}, \tag{2.6}$$

where y_k is the probability of the expected target label, y_c is the probability of label c and C is the set of all labels. The cross-entropy loss has an efficient GPU implementation in machine learning libraries. Other popular loss functions include the binary cross-entropy loss for binary classification and mean-squared error (MSE) for regression.

In the backward pass the network computes the gradient of the loss with respect to each weight using backpropagation. This involves applying the chain rule to propagate errors backward from the output layer to the input layer. Unlike earlier learning paradigms, backpropagation allows the update of all parameters at once, it is also called *end-to-end* training.

The weights and biases are updated using an optimization algorithm, typically Gradient Descent or its variants (e.g., Adam (Kingma and Ba, 2014)). The update reduces the loss function by moving the weights in the direction of decreasing error, with the learning rate determining the step size. As GPUs are particularly suited for parallel processing, these steps are generally done on multiple data samples or *minibatches* at a time to save computational resources. Batch training also reduces the variance of the gradient signal since the backpropagated gradients are averaged over multiple samples. An *epoch* is a complete pass over the dataset. The training process is repeated for a certain number of epochs or it is stopped via *early stopping*. Early stopping is applied when we no longer see improvement on the validation dataset.

The size of the batches, the parameters of early stopping along with predefined architectural choices such as the number of layers and neurons are considered *hyperparameters*. Hyperparameters need to be defined before the forward pass of backpropagation. Hyperparameter optimization is a difficult problem and sophisticated algorithms such as genetic algorithms often perform no better than simple heuristics.

Training neural networks can be challenging due to several issues, one of the most common being the vanishing and exploding gradient problem. As information is propagated backward through deep networks during backpropagation, gradients can either shrink or grow exponentially. Vanishing gradients make it hard for weights in early layers to be updated, resulting in poor learning for those layers, while exploding gradients cause the weights to grow uncontrollably, leading to instability. ReLU helps mitigate the vanishing gradient problem by avoiding very small gradients. In addition, techniques like gradient clipping can be applied to keep gradients within a certain range, preventing them from growing too large.

Another challenge in training neural networks is overfitting, where the model performs well on the training data but fails to generalize to unseen data. This happens when the network becomes too complex, memorizing the training examples rather than learning the underlying patterns. Several techniques are used to combat overfitting. One popular method is *dropout*, which randomly zeroes a subset of neurons during each training iteration, preventing the network from relying too heavily on any single neuron. Regularization techniques like L_2 regularization (weight decay) also help by penalizing large weights, encouraging the model to learn simpler, more generalizable representations.

2.2.3 Sequence modeling

Virtually all problems in NLP deal with sequences of symbols. These symbols can be characters, subwords, words or even sentences. Sequences can be of arbitrary length. Natural language often uses long-range dependencies which poses another difficulty to modeling.

Sequence modeling has a long history predating the current surge of deep learning and previous sequence modeling techniques, such as Hidden Markov Models and Finite State Automata (FSA) – which were and are still widely used for computational morphology – show remarkable success in this domain. These models were not differentiable and the model parameters were derived via iterative maximum likelihood methods such as expectation maximization. Rule based systems also enjoyed considerable success but they relied on extensive work from linguistic experts. There were recent attempts to restrict these powerful modeling techniques in ways that make them differentiable thus allowing end-to-end training with backpropagation on labeled data. Chapter 4 explores one such application in morphology, a restricted FSA, originally used for sequence classification.

Sequence modeling techniques with neural networks date back to the 1960s with Rosenblatt's addition of loops to MLPs (Rosenblatt, 1957). In the next sections we give a quick overview of the main building blocks of sequence modeling neural networks.

2.2.3.1 RNN

Recurrent Neural Networks (RNNs) are a class of neural networks designed to handle sequential data, where the order of the inputs matters. Unlike feedforward networks, RNNs have connections that form loops, allowing them to maintain a memory of previous inputs. This makes them particularly effective for tasks where context is important, such as NLP, time-series prediction, and speech recognition. The general formulation of RNNs is:[1]

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.7}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{2.8}$$

where x_t is the input vector at timestep t, h_t is the *hidden state*, y_t is the output vector, σ_h and σ_y are activation functions. The trainable parameters are W_h, W_y and U_h projection matrices and b_h and b_y biases. The hidden state is updated every timestep based on the previous hidden state and the input. The output is dependent on the current hidden state. The key feature of RNNs is their ability to process sequences of variable lengths since the hidden state is updated with every new input x_t. This recurrent structure enables RNNs to capture temporal dependencies.

Since natural sequences are finite, the cyclic graphs of RNNS can be *unrolled in time*. The resulting acyclic graph is now trainable with backpropagation, also called *backpropagation through time* or *BPTT*. However, this graph grows linearly in size with the length of the sequence and basic RNNs face challenges like the vanishing and exploding gradient problem, making it difficult to learn long-term dependencies. To address this, more advanced versions like *Long Short-Term Memory (LSTM)* (Hochreiter and Schmidhuber, 1997) networks and *Gated Recurrent Units (GRU)* (Cho et al., 2014) were developed.

2.2.3.2 LSTM

Long Short-Term Memory (LSTM) (Figure 2.2) networks are a type of RNN architecture introduced by Hochreiter and Schmidhuber (1997). LSTMs are designed to handle the vanishing gradient problem, which standard RNNs face when learning long-term dependencies in sequential data. LSTMs achieve this by incorporating *memory cells* that can maintain information over long time steps. An LSTM unit consists of three main components: *forget gate*, *input gate*, and *output gate*. The forward pass of a standard LSTM is defined as:

[1]This is the Elman network, one of the many similar RNN formulations.

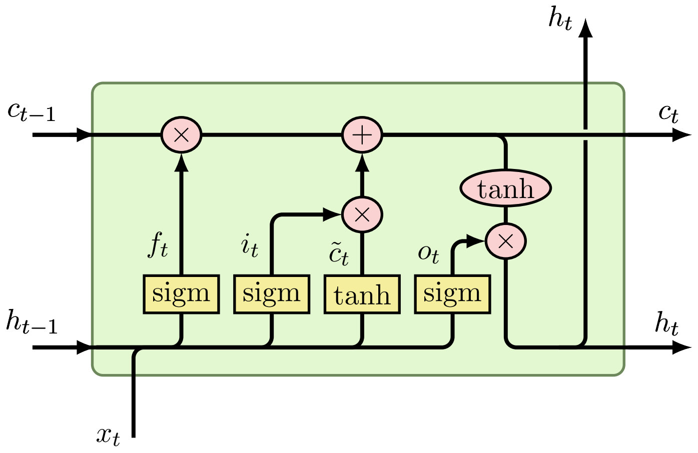

Figure 2.2: Basic structure of an LSTM cell. Figure adapted from https://github.com/mvoelk/nn_graphics.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.9}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.10}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.11}$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.12}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.13}$$
$$h_t = o_t \odot \sigma_h(c_t). \tag{2.14}$$

Here, f_t is the forget gate, which decides which part of the previous memory to forget. The input gate i_t determines which values from the input should update the memory cell. The output gate o_t determines the output and the next hidden state (Equation 2.14). The outputs of all three gates are dependent only on the input at time t and the previous hidden state h_{t-1}. The sigmoid function squashes their output between 0 and 1. The cell state (separate from the hidden state) is updated in two steps (2.12 and 2.13) and finally the next hidden state is updated based on c_t and the output gate o_t. This gating mechanism allows LSTMs to capture long-term dependencies by carefully controlling the flow of information over time. A similar but slightly simpler recurrent cell is the *Gated Recurrent Unit* or GRU (Cho et al., 2014).

2.2.3.3 Sequence modeling with LSTMs

LSTMs are rarely used as individual cells; instead, both the cell state and hidden state are represented as real-valued vectors. The input to LSTMs is also typically real-valued vectors, often in the form of word embeddings. While embeddings are discussed in more detail in Section 5.1.1, for now, they can be understood as static mappings from words to real-valued vectors. Figure 2.3 provides a basic example of an LSTM being applied to sentiment analysis, a common sequence classification task. Since LSTMs process sequences in only one direction, it is common practice to pair them with another LSTM that processes the sequence in the opposite direction, forming a bidirectional LSTM (as opposed to a unidirectional LSTM). The final output of both LSTMs is usually concatenated and used as a representation for the full sequence. LSTMs can also be stacked into multilayer LSTMs where the output of one LSTM is the input of the next layer of LSTMs. Finally one or more dense layers may be used to project into the desired dimension.

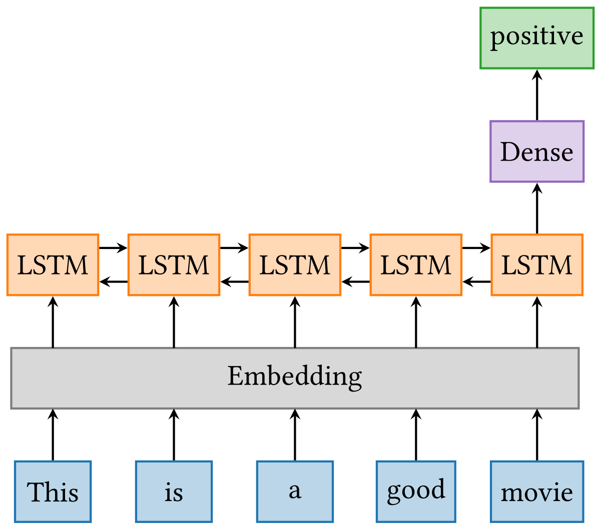

Figure 2.3: Sequence classification example with one layer LSTM.

Sequence tagging or *sequence labeling*[2] is a set of tasks where one label is assigned to each unit of the text, usually one for each token. Examples include part-of-speech (POS) tagging and named entity recognition (NER). The LSTM architecture depicted in Figure 2.4 is identical to the one used for sequence classification but we use every output of the LSTM not just the last one.

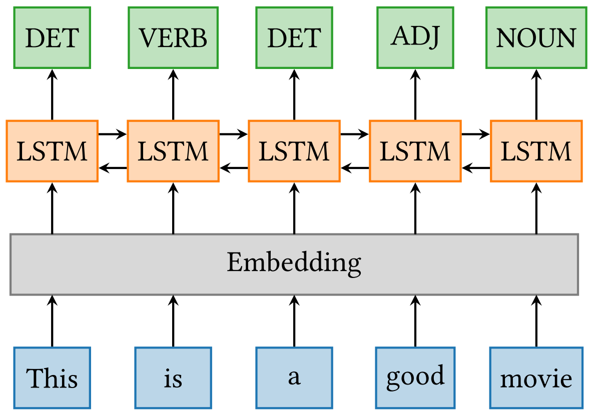

Figure 2.4: Sequence tagging example.

2.2.3.4 Encoder-decoder model

Sequence classification and tagging required a fixed output length (a single one of one for each token) but there are many problems in NLP where the length of the output is different from the length of the input. Machine translation is one such example where an input sentence in one language is usually translated into a different number of words in another language. The *encoder-decoder* or *sequence-to-sequence* (seq2seq) architecture provides a general solution to this problem (Sutskever et al., 2014).

[2]Sometimes called token classification.

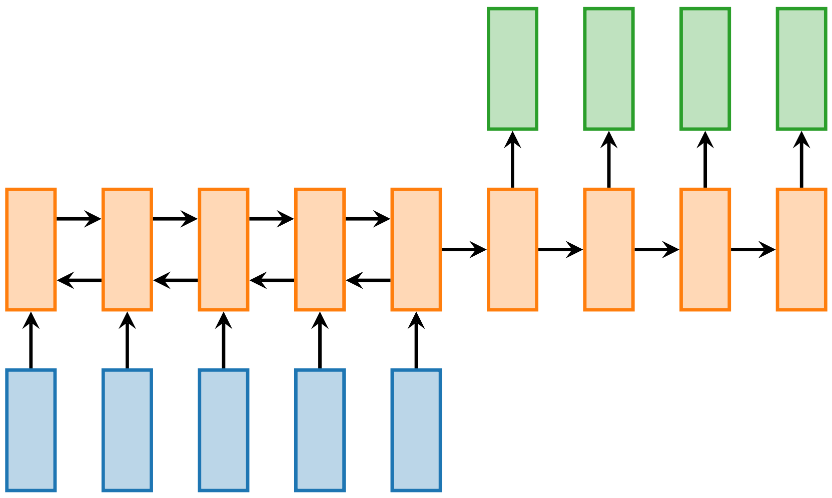

Figure 2.5: Encoder-decoder

The general idea is illustrated in Figure 2.5. The input, usually depicted at the bottom, is first encoded into a single vector regardless of its length. The encoder can be a bidirectional LSTM. The representation of the input is then used to initialize the decoder which generates the output one symbol at a time from left to right. Each generated symbol is conditioned on both the input and the previously generated symbols. The generation is terminated when the decoder outputs a predefined end-of-sequence symbol.

While the encoder-decoder model can map arbitrary sequences and even between different domains, it struggles with long sequences and long-range dependencies. The encoder squashes the input into a fixed size vector regardless of its length and this proved to be a serious information bottleneck. The *attention mechanism* (Bahdanau et al., 2014; Luong et al., 2015) alleviates this problem by enabling the decoder to selectively focus on certain parts of the input as depicted in Figure 2.6. The main idea of attention is the for each decoding step, the decoder looks at all the outputs of the encoder and computes a weighted sum of the intermediary output vectors called the *context vector*. The next output symbol is conditioned on this context vector. A new context vector is computed for the next output symbol. Attention has various versions with very similar effectiveness.

2.2.3.5 Transformer

Although LSTMs with attention deal well with mid-range dependencies, they have serious limitations when it comes to long-term dependencies and parallelization. This is because LSTMs process sequences step by step, making them inherently sequential and slowing down the training process, especially for long sequences. To address these challenges, Transformers (Vaswani et al., 2017) revolutionized sequence processing by eliminating the need for recurrent structures (horizontal arrows in Figure 2.3) altogether. Transformers rely entirely on *self-attention mechanisms*, which allow them to process all elements of a sequence simultaneously.

The Transformer architecture is illustrated in Figure 2.7. The encoder (left side) first tokenizes the source sequence. Based on the application, there are different choices for tokenization which we explore in Section 5.3. The tokens are encoded with a static embedding. *Positional encoding* replaces the role of recurrent connections by adding a fixed vector to the embedding vector based on the relative position of a token in the sequence. Positional encoding vectors are defined as:

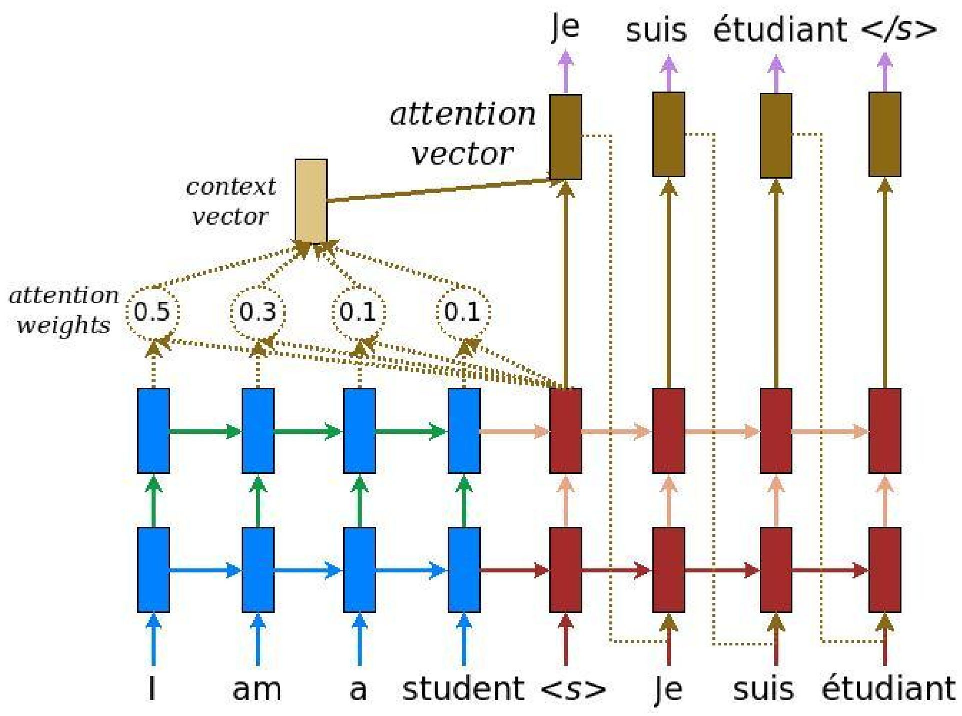

Figure 2.6: Attention mechanism example. Source url.

$$PE_{(\mathrm{pos},2i)} = \sin(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}) \tag{2.15}$$

$$PE_{(\mathrm{pos},2i+1)} = \cos(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}), \tag{2.16}$$

where pos is the relative position starting from 0, i is the ith dimension of the vector and d_{model} is the size of the embedding vector. The next step is the actual self-attention operation on the embedding (plus positional) vectors. First, it computes the *key*, the *query* and the *value* vectors (K, Q and V) by multiplying the input vectors with learned projection matrices. It then performs a so-called *scaled dot-product attention* defined as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V. \tag{2.17}$$

The output of self attention is a weighted sum of the value vectors for each input vector. The weights are computed using the query and the key vectors and the softmax function normalizes it. This is done simultaneously for every input as matrix operations. One set of key, query and value parameters is called a single attention head but it is seldom used alone. Multi-headed attention involves dozens of attention heads with the outputs concatenated. This is repeated for multiple layers where the output of one layer is the input of the next layer. There is a normalization layer between each step and the output is passed through a feed-forward network with ReLU activation.

The decoder works very similarly with two notable differences. In addition to self attention on the generated output sequence, it also performs attention on the input sequence called *cross-attention*.

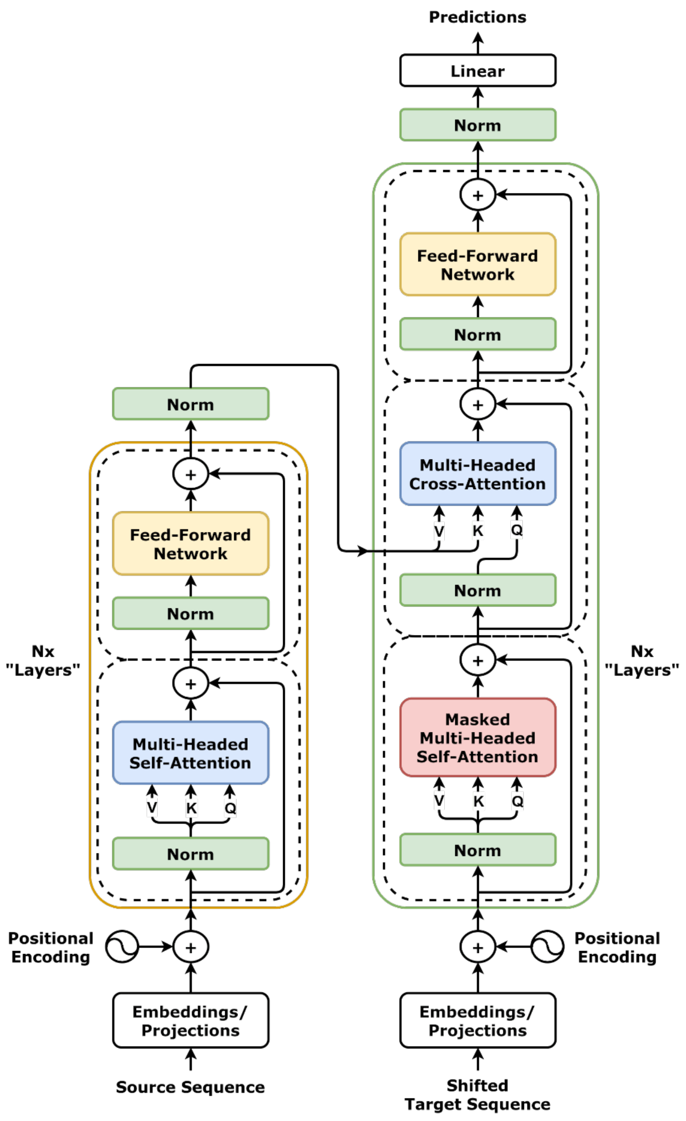

Figure 2.7: Transformer architecture. Source: Wikipedia

The other difference is the use of masking. Similarly to LSTM-based seq2seq models, the output is generated one token at a time during inference, therefore the training process should not expose *future* symbols to the decoder. During the generation of token t, all tokens right of t are masked out from the self attention operation.

Transformers are notoriously harder to train than LSTMs but certain *tricks* make training more stable. The most important is layer normalization after each sublayer ($\text{LayerNorm}(x + \text{Sublayer}(x))$). Another one is the use of learning rate warmup.

Transformers are highly parallelizable due to the lack of recurrent connections which paved the way for an explosion in model size. Vaswani et al. (2017) used 6 to 8 layers, up to 1024 dimensional embeddings and 8 attention heads in the original paper. Transformer-based models have come a long way since then and as of 2024 the largest models use over 50 layers, 100+ attention heads and large embeddings with over 10,000 dimensions. As of November 2024, the largest openly available model is LLaMA 2 with 70 billion parameters (Touvron et al., 2023). Part II of this thesis describes a large-scale evaluation of the highly successful generation of Transformer models predating the current explosion of LLMs.

2.3 Introduction to morphology

Morphology is the study of internal word structure and word formation. The basic units of discussion are *morphemes*, the smallest linguistic units with independent meaning or grammatical function. Words consist of morphemes. Morphemes can be classified as *root* or *affix*. Root morphemes are the nucleus of the word and affixes attach to root morphemes. A similar categorization is *free* vs. *bound* morphemes. Free morphemes can appear as words by themselves while bound morphemes must attach to free morphemes. This distinction is language dependent, for example the past tense morpheme is bound in English but free in Chinese [3].

Morphological processes are generally divided into two types: *derivation* and *inflection*. Derivation creates *new* words, often changing the part of speech or the grammatical category of the root word. English examples of derivational morphology include *-ly, -ness* and *-able*. Inflection on the other hand creates different forms of the same word. *Lemmas* are the canonical or dictionary entries of words and the set of all forms of the same lemma are called *lexemes*. For example, *go, went, gone* and *goes* belong the same lexeme of which the lemma is *go*.

2.3.1 Morphological typology

Morphological typology is a subfield of linguistic typology which categorizes the languages of the world according to their dominant word formation processes, i.e. the way they combine morphemes into words. The two main categories are analytic vs. synthetic languages. *Analytic* languages feature little to no inflection and most information is carried in auxiliary words and word order. Conversely, *synthetic* languages use inflection to change the meaning of a word. At the extreme end of this scale are *polysynthetic* languages which can combine the meaning of full sentences in single words. Synthetic languages are further divided into *agglutinative* and *fusional* languages. Agglutinative languages rely on affixes, usually prefixes and suffixes. Multiple affixes can be combined by concatenating them and the root often remains easily extractible. This is not the case for fusional languages where multiple morphemes are combined into one morpheme in a way that the original morphemes, and often the root word, are not segmentable. An interesting subtype of fusional morphology is *templatic morphology*, where root words are templates with slots to be filled by affixes. It is important to note that natural languages often do not fall clearly into one category and they may exhibit characteristics of more than one morphosyntactic system. For example English is a largely analytic language with heavy reliance on auxiliary verbs and pre- and postpositions but it retains some of the fusional elements of its Germanic heritage in its verbal morphology. Hungarian, while largely agglutinative, has some fusional elements as well (cf. Section 2.3.2).

While each language has its own characteristics, related languages tend to have exhibit similarities. The largest example of analytic languages is the Sino-Tibetan family which includes every Chinese dialect. Most Indo-European languages are fusional with the notable exception of English. However most of the world's languages are agglutinative including Hungarian and most of the Uralic family.

[3] https://ufal.mff.cuni.cz/~hana/teaching/ling1/05-Morphology.pdf

Although we deal with over 50 languages in this thesis, not all types of morphosyntactic systems are equally represented. Our main source of data is the Universal Treebank which is only available in sufficient quantity in European languages. While the Indo-European family already represents a large variety of typological features (Skirgård et al., 2023), there are some missing types such as polysynthetic languages. Unfortunately polysynthetic languages are all low resource and they lack sufficient labeled data for our purposes. Furthermore our only examples of templatic morphology are Arabic and Hebrew. We do study a large variety of fusional and agglutinative (mostly Uralic and Indo-European) languages.

2.3.2 Hungarian morphology

We pay special attention to Hungarian throughout this thesis. Aside from Hungarian being the native language of the author and many of her coauthors, it provides an interesting case study in Chapter 4 and the main topic of Section 7.1.

Hungarian is an agglutinative language which mainly uses a large variety of suffixes (for a comprehensive summary of inflectional suffixes see Veenker (1968)) and one circumfix for superlatives (*leg- ADJ -bb*).

Nominal morphology, the morphology of nouns and pronouns, features a large number of cases with the addition of plural and possessive suffixes resulting in hundreds of inflectional forms for each lemma. The order of the suffixes is fixed and the phonological changes at the morpheme boundaries are highly regular. *Verbal* morphology exhibits similar processes along with some fusional elements.

The most prominent of these processes is *vowel harmony*. Most suffixes are in fact templates such *bVn*, the morpheme for the inessive case, meaning *in, inside*, where the vowel is filled based on the phonological class of the vowels of the stem. *bVn* can be realized as either *ban* or *ben* depending the front or backness of the vowels in the stem. Many other morphemes have two forms, front and back, while some have three forms (front unrounded, front rounded and back) and some have four (Siptár and Törkenczy, 2007). Vowel harmony is not unique to Hungarian. Finnish, its distant relative, has a simpler form of vowel harmony. Although modern Estonian no longer uses vowel harmony, many other Uralic languages do. Among the languages we examine in Chapter 6, Finnish, Turkish and Persian have vowel harmony, while Korean has some limited form of vowel harmony.

Part I

Deep Learning for Morphology

Chapter 3

Morphological Inflection and Analysis

Morphological inflection is the task of inflecting a lemma given either a target form or some contextual information. Analysis is the opposite, an inflected form is the input and its morphological analysis is the desired output. Morphology has traditionally been solved by finite state transducers (FST) that employ a large number of handcrafted rules. The discrete nature of such processes makes it difficult to directly translate transducers into neural networks and to effectively train them using backpropagation. There have been various attempts to replace parts of the FST paradigm with neural networks (Aharoni and Goldberg, 2016).

In this chapter we apply various kinds of neural encoder-decoder models and we demonstrate their effectiveness on both analysis and inflection. We first showcase a series of proof-of-concept experiments on Hungarian morphological analysis with three types of encoder-decoder models, two of them offering some level of interpretability (Section 3.1). We then introduce our submission to the 2018 SIGMORPHON Shared Task which involved type-level inflection in 100 languages and in-context inflection in 7 languages (Section 3.2).

3.1 Hungarian morphological analysis

Finite state transducers are highly successful in modeling Hungarian but the creation of such rule based systems requires tremendous expert work. The latest such system is emMorph (Novák et al., 2016) which builds on previous systems and uses a tagging schema designed by a team of Hungarian linguists (Rebrus et al., 2012; Novák et al., 2017). emMorph uses the HFST engine (Lindén et al., 2011). The resulting analyzer is fast, interpretable and highly accurate but it is unable to deal with out-of-vocabulary words such as neologisms from English. In this section our main question is to what degree we can imitate emMorph with encoder-decoder models.

Encoder-decoder models have been demonstrated to work very well for morphological inflection (Cotterell et al., 2016; Kann and Schütze, 2016a; Kann et al., 2016) across a wide range of typologically rich languages including Hungarian. However, the training data for these competitions was harvested from Wiktionary inflection tables which aim to include every possible inflected form of a lemma regardless of its frequency. We remedy this by using the word forms and their analyses form the Szeged Korpusz (Csendes et al., 2005; Vincze et al., 2010).

emMorph outputs two analyses for a token on average. We do not try to disambiguate them and we accept any of the possible analyses as correct from our encoder-decoder networks.

3.1.1 Data

The data is a filtered version of the Szeged Korpusz. It contains 159,131 unique token-analysis pairs. emMorph is unable to analyze 5.6% of these which we exclude from our training data but we do run inference on them (cf. Section 3.1.4). We are left with 63,314 unique lemmas which we split into three sets, 58,314 train, 2,500 development and 2,500 test lemmas. We then split the inflected forms along with their lemmas. We end up with 139,016 train, 5,584 development and 5,292 test word types. We include every possible analysis of every form as separate samples resulting in 279,440 train, 11,724 development and 10,284 test samples in total. 92.1% of the tokens are nouns (54.7%), verbs (21%), or adjectives (16.4%). There are 10 smaller POS classes which we group together when analyzing the results. Hard attention models require step symbols in the target side of the data. We add a step symbol whenever a source character is consumed by the transducer.

3.1.2 Models

We use three types of models in this section. The first one is a "standard" LSTM encoder-decoder with Luong attention (cf. Section 2.2.3.4). The second one is an identical model with hard monotonic attention. *Hard attention* is a type of attention where the attention weights are concentrated on exactly one symbol instead of a *soft* distribution looking at the full source sequence. *Monotonic* refers to the other restriction whereas the attention head can only move from left to right when generating a new symbol. Effectively this means that after each generation step, the attention head either remains on the same source symbol or moves ahead one step. This is controlled with artificial step symbols in the training data. The third model is a SoPa encoder-LSTM decoder model which we introduce in more detail in Chapter 4.

We experimented with various hyperparameters but it turns out that the models are not very sensitive to the hyperparameters as long as the there is sufficient capacity i.e. enough trainable parameters. We set the hyperparameters the same for the three models when possible. The LSTMs all used a single layer and their hidden size was set to 512. We used separate embeddings for the encoder and decoder side of the models both set to 32 dimensions. The vocabulary consisted of characters and morphosyntactic tags. The standard attention and hard monotonic attention models have no extra hyperparameters. In the case of SoPa Seq2seq pattern length and the number of patterns are hyperparameters. After some initial testing, we used pattern lengths from 3 to 7, 25 apiece.

We use identical training parameters for the three types of models. We train the models with the Adam optimizer with with $\mathrm{lr} = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$. The batch size is always set to either 128 or 256 and we use early stopping based on the development accuracy. The number of trainable parameters is 4.3M for the Luong attention model, 4.6M for the hard attention model and 6.3M for SoPa Seq2seq.

3.1.3 Results

Figure 3.1 shows the accuracy of each model by POS class.. Standard attention is the best out of the three models when it comes to pure accuracy (the output is identical to either analysis of emMorph). This is true in all POS classes. Hard monotonic attention is a close second and SoPa Seq2seq is the third with a noticeable gap. Still, SoPa Seq2seq is over 75% correct for all major POS.

Our results are noticeably worse than the best SIGMORPHON submissions on Hungarian which may be due to two reasons. First, we use a more detailed tagging scheme than the schema used by SIGMORPHON (UniMorph). emMorph splits words into morphemes and analyzes them separately. It also outputs multiple possible analyses. Second, the SIGMORPHON shared tasks use type level data

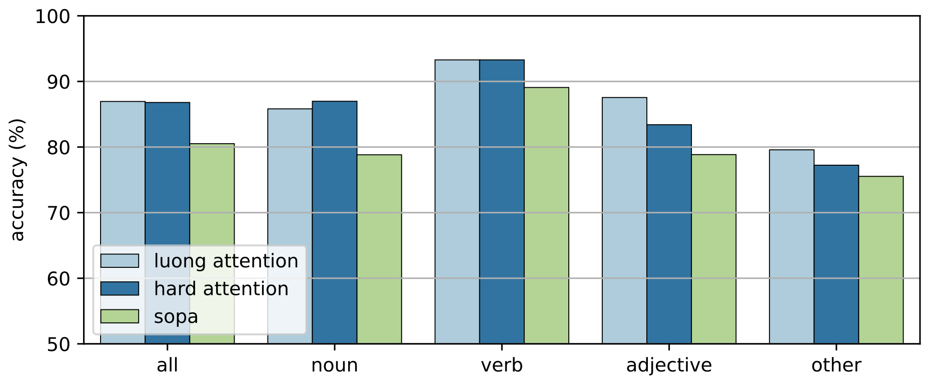

Figure 3.1: Morphological analysis results by model and POS.

with very rare forms having the same weight in evaluation as frequent forms. Less frequent words tend to be more regular making the task easier for neural models.

Both hard monotonic attention models and SoPa Seq2seq offer some form of interpretability. Chapter 4 explores the interpretability of SoPa Seq2seq models in 9 languages including Hungarian. Hard monotonic attention models are trained with step symbols in the target string so they output step symbols during inference as well. When a step symbol is generated, the attention head moves one symbol to the right and uses the next input symbol. This mechanism is similar to transducers albeit much simpler. When we manually inspect some examples from the test set, we find that they are linguistically plausible and they often coincide with expert analysis.

3.1.4 Analyzing out-of-vocabulary words

emMorph is unable to analyze words where the stem is not recognized or impossible to determine. Out-of-vocabulary (OOV) words account to roughly 5% of word types. The most common reason for OOV words are misspellings, unusual compounds, foreign words and slang. One advantage of neural models is that they can take any input. Since the vocabulary is the set of characters, arbitrary character symbols are accepted as inputs. If the input contains out-of-vocabulary characters (or words in word-based models), we replace them with a dedicated *unknown* symbol. We ran the neural model on the 5% OOV tokens. Table 3.1 shows a few selected examples of OOV words and their analysis by each model.

Input	Translation	Standard attention	Hard attention	SoPa
ópiumosdobozból	from opium box	ópiumos[/N] doboz[/N]ból[Ela]	ópium[/N]os[_Adjz:s/Adj] doboz[/N]ból[Ela]	ópiumosdoboz[/N] ból[Ela]
jutatni	to send (mistyped, correct: juttatni)	jut[/V] at[_Caus/V]ni[Inf]	jutat[/V]ni[Inf]	jutat[/V]ni[Inf]
vólna	was (archaic form)	vól[/V] na[Cond.NDef.3Sg]	vól[/V] na[Cond.NDef.3Sg]	válik[/V] na[Cond.NDef.3Sg]
Windowsszal	with Windows	Windows[/N]zal[Ins]	Windows[/N]al[Ins]	Windowsal[Ins]

Table 3.1: Examples of OOV words and their analysis. Newlines were inserted for readability.

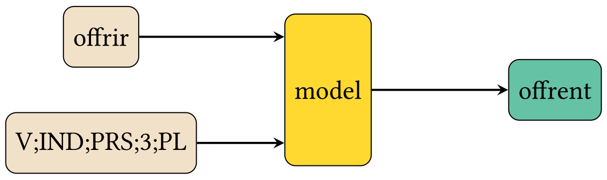

Figure 3.2: Inputs and output for the model

3.2 The Universal Morphological Reinflection Shared Task

SIGMORPHON first organized a shared task on morphological inflection in 2016 (Cotterell et al., 2016) which involved both inflection (inflect a word given its lemma) and reinflection (inflect a word given another inflected form of the same lemma). The winning solution (Kann and Schütze, 2016b) used a character sequence-to-sequence network with Bahdanau's attention (Bahdanau et al., 2015). Kann and Schütze (2016b) and other neural solutions (Aharoni et al., 2016) were up to 10% point better than rule-based and transducer-based solutions, resulting in the wholesale abandonment of the rule-based paradigm for morphology. In the second edition of the shared task (Cotterell et al., 2017) most teams used similar settings.

In this section I present my submission as an individual team for the 2018 CoNLL–SIGMORPHON Shared Task: Universal Morphological Inflection (Cotterell et al., 2018). My system placed 2nd in both tasks and all tracks (Ács, 2018).

3.2.1 Task formulation

In this section we briefly describe the objective of the task and provide examples for each subtask. A more comprehensive explanation is available on the shared task's website[1] and in task description paper (Cotterell et al., 2018).

3.2.1.1 Task1: Type-level inflection

Inflection aims to find an inflected word given its lemma and a set of morphological tags in UniMorph (Kirov et al., 2018). The process is illustrated in Figure 3.2 and a few examples are shown below (the second column is the target):

release	releasing	V;V.PTCP;PRS
deodourize	deodourize	V;NFIN
outdance	outdancing	V;V.PTCP;PRS
misrepute	misrepute	V;NFIN
vanquish	vanquished	V;PST
resterilize	resterilizes	V;3;SG;PRS

The shared task featured over 100 languages and 10 additional surprise languages were released before the submission deadline. Most languages had three data settings: high (10,000 samples), medium (1,000 samples) and low (100 samples), except some low-resource languages that did not have

[1]https://sigmorphon.github.io/sharedtasks/2018/

enough samples for high or medium settings. Each language had a development set of 1,000 or fewer samples.

3.2.1.2 Task2: Inflection in context

Task2 is a cloze task. We were given a sentence with a number of missing word forms (usually 1 or 2) and our task was to inflect the word given its lemma and context. Task2 two had two tracks: in Track1 all the lemmas and morphosyntactic description are given in the sentence context (the morphosyntactic description of the covered word is covered too), and in Track2 only the word forms of the context are given. Below are examples from Track1 (the missing form is highlighted):

Les	le	DET;DEF;FEM;PL
compagnies	compagnie	N;FEM;PL
aériennes	aérien	ADJ;FEM;PL
à	à	ADP
bas	bas	ADJ;MASC;SG
coût	coût	N;MASC;SG
ne	ne	ADV;NEG
_	connaître	_
pas	pas	ADV;NEG
la	le	DET;DEF;FEM;SG
crise	crise	N;FEM;SG

and the same sentence for Track2:

Les	_	_
compagnies	_	_
aériennes	_	_
à	_	_
bas	_	_
coût	_	_
ne	_	_
_	connaître	_
pas	_	_
la	_	_
crise	_	_
.	_	_

Both examples are taken from the development sets. The training sets have no covered words, and we generated training examples by covering a single word at a time, and using the rest as its sentence context.

Task2 also featured low, medium and high resource settings with roughly 1,000, 10,000 and 100,000 tokens respectively.

3.2.2 Task1 model: two-headed attention

In this section we describe our system for Task 1: Type-level inflection. We explain our experimental setup and the random hyperparameter search, and finally we list three slightly different submissions and their results.

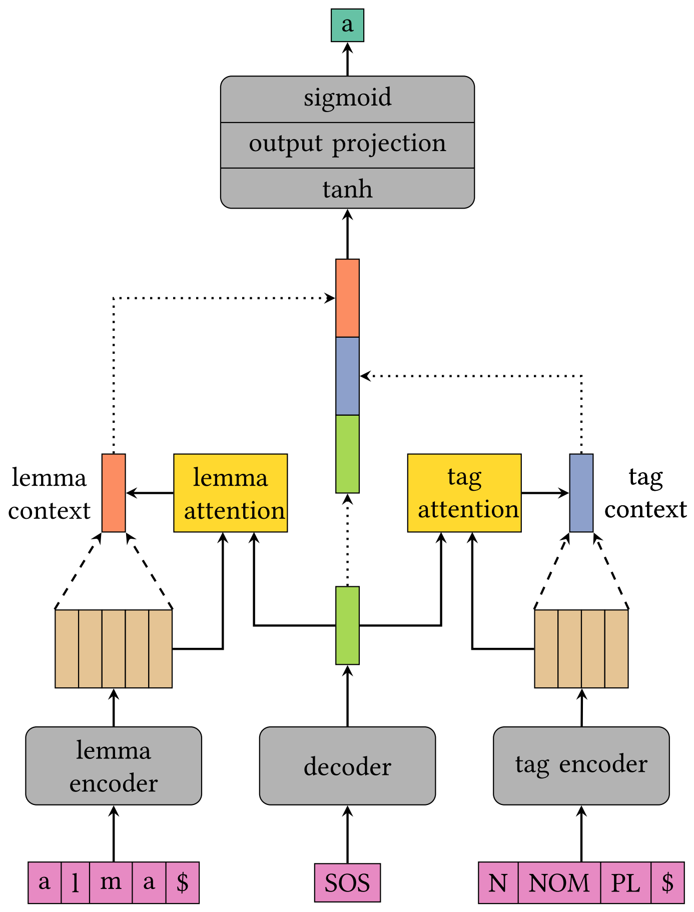

Figure 3.3: Two-headed attention model used for Task1. The figure illustrates the first timestep of decoding. The output of this step is fed back to the decoder in the next timestep. Modules are colored gray, attention heads yellow, inputs are purple, outputs are teal and encoder output matrices are salmon. Dotted arrows represent copy operations and dashed arrows represent attention summaries.

3.2.2.1 Two-headed attention seq2seq

Inflection can be formulated as a mapping of two sequences, namely a lemma and a sequence of tags, to one sequence, the inflected word form. The lemma and the inflected word forms are character sequences that usually share a common alphabet while the tags are a sequence of language-specific morphosyntactic codes. Figure 3.3 illustrates our architecture. We use separate encoders for the lemma and the morphological tags and a single decoder. Both encoders employ character/tag embeddings and bidirectional LSTMs, where the outputs are summed over the two directions. The two encoders' hidden states are then linearly projected to the decoder's hidden dimension and used to initialize the decoder's hidden state. This allows using different hidden dimensions in each module. Decoding is done in an autoregressive fashion, one character at a time. At each timestep the decoder reads a single character: *SOS (start-of-sequence)* at first, the ground truth during training (*teacher forcing*) and the previous output during inference. The decoder uses a character level embedding, which may or may not be shared with the lemma encoder (cf. Section 3.2.2.2), then it passes the embedded symbol to a unidirectional LSTM. Its output is used by two attention modules, hence the name *two-headed attention*, to compute a context vector using Luong's attention (Luong et al., 2015). The lemma and tag context vectors are concatenated with the decoder output, then passed through a *tanh*, an output projection and finally a sigmoid layer which produces a distribution over the character vocabulary of the language. Greedy decoding is used.

3.2.2.2 Experimental setup

We created our own experiment framework that allows running and logging a large number of experiments. The framework is available on GitHub[2] and the configurations and scripts used for this shared task are available in a separate repository[3]. The latter repository contains all best configurations including the random seeds (we generate the random seeds at the beginning of each experiments, then save them for reproducibility).

All experiments shared a number of configuration options while the others were randomly optimized. We list the ones we fixed here and the others in Section 3.2.2.3. Each experiment used a batch size of 128 for both training and evaluation except the ones on the Kurmanji language because the development dataset contained very long sequences and we had to reduce the batch size to 16 to fit into memory (12GB). We used the Adam optimizer with learning rate 0.001 and we stopped each experiment when the development loss did not decrease on average in the last 5 epochs compared to the previous 5 epochs. We ran at least 20 epochs before stopping even if the early stopping condition was satisfied to avoid early overfitting, which happened in about 10% of the experiments. We also set a hard upper limit for the number of epochs (200) but this was reached only two times out of 1,886 experiments. The average number of epochs before reaching the early stopping condition was 51 and only 2.7% of experiments ran for more than 100 epochs. After each epoch, we saved the model if its development loss was lower than the previous minimum. We used cross entropy as the loss function.

3.2.2.3 Random parameter search

Our initial experiments suggested that the model is very sensitive to random initialization and the same configuration can result in models with very different performance. This is probably due to the limited training data even in *high* setting and the large number of parameters of the model. We chose three languages, Breton, Latin and Lithuanian, and ran a large number of experiments with random configuration on them. The reason these were chosen is that the development accuracy on these were in the mid-ranges among all the language during our initial experiments. The following random experiments were all run on the *high* training sets. Common parameters (cf. Section 3.2.2.2) were loaded from a base configuration and some parameters were overridden with a value uniformly sampled from a predefined set. The range of values are listed in Table 3.2. Both encoders (lemma and tag) and the decoder (listed as *inflected*) have three varying parameters: the size of the embedding, the number of hidden LSTM cells and the number of LSTM layers. We also varied the dropout rate for both the embedding and the LSTMs and the whether to share the vocabulary and the embedding among the lemma and the decoder or not.

The running time of an experiment is dependent on the average length of the input sequences and the size of the vocabulary. It turns out that these vary greatly among the languages in the dataset. As listed in Table 3.3 Breton is much ‘smaller’ in both alphabet and sequence length than Lithuanian or Latin and this was evident from the difference in average running time.

Table 3.4 summarizes the results of our random parameter search. Since the average running time of different language experiments is very different, we ended up running many more Breton

[2]https://github.com/juditacs/deep-morphology

[3]https://github.com/juditacs/sigmorphon2018

Parameter	**Values**
dropout	0.1, 0.3, 0.4, 0.6
share vocab	true, false
inflected embedding size	10, 20, 30, 50
inflected hidden size	128, 256, 512, 1024
inflected num layers	1, 2
lemma embedding size	10, 20, 30, 50
lemma hidden size	128, 256, 512, 1024
lemma num layers	1, 2, 3, 4
tag embedding size	5, 10, 20
tag hidden size	64, 128, 256
tag num layers	1, 2, 3, 4

Table 3.2: Parameter ranges used for the random hyperparameter search.

experiments in roughly the same time. The standard deviation of results is quite large, especially for Breton, which we attribute to the small alphabet, the short sequences and the small number of lemmas (44) as opposed to Latin (6517) or Lithuanian (1443).

	Breton	**Latin**	**Lithuanian**
alphabet size	27	55	58
inflected maxlen	14	23	32
inflected types	1,790	9,896	9,463
lemma maxlen	11	19	28
lemma types	44	6,517	1,443
tag types	20	33	34
tags maxlen	9	7	6

Table 3.3: Dataset statistics of the three languages we used for hyperparameter optimization.

We observed that models with the same parameters often result in very different word accuracy. To test this, we took the best performing configuration for each language and trained 20 models by language with identical parameters but different random seeds. Table 3.5 and Figure 3.4 show that identical parameters can result in models with very different performance.

		Breton	**Latin**	**Lithuanian**
experiments		1033	610	243
	mean	70.92	62.32	80.25
dev acc	max	93.00	78.90	88.40
	std	28.70	11.30	8.37
time	mean	0.83	5.42	8.61

Table 3.4: Summary of the parameter search. The running time is given in minutes.

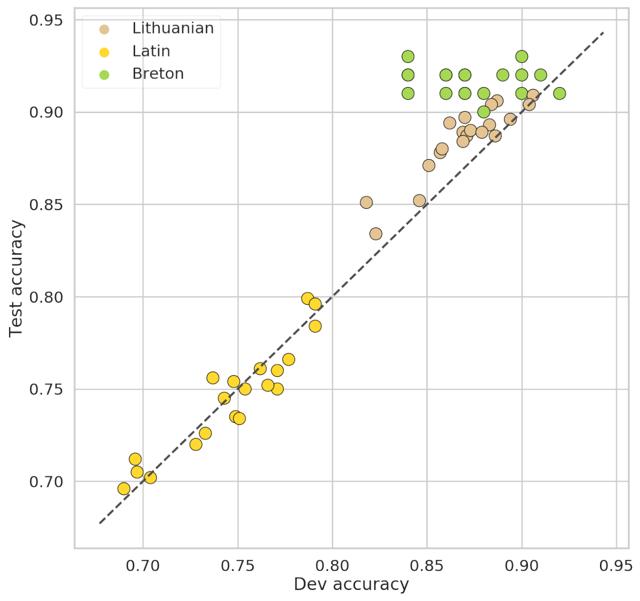

Figure 3.4: The test and development accuracy of identical models initialized with different random seeds.

		Breton	Latin	Lithuanian
train acc	mean	96.29	92.58	96.69
	std	1.39	3.21	2.25
	min	94.30	84.57	90.51
	max	99.04	97.14	99.08
dev acc	mean	87.35	74.73	86.95
	std	2.41	3.17	2.32
	min	84.00	69.00	81.80
	max	92.00	79.10	90.60

Table 3.5: Accuracy statistics of 20 models trained with the same parameters but different random seed.

3.2.2.4 Submission

We took the 5 highest scoring model for Breton, Latin and Lithuanian, our languages used for hyper-parameter optimization and trained a model with those parameters for each language and each data size, thus training 15 models per language and data size. Our first submission is simply the model with the highest development word accuracy. The second submission is the result of majority voting by all 15 models. The third one is the same as the first one but we changed the evaluation batch size from 128 to 16. This results fewer pad symbols on average. Table 3.6 lists the mean performance of each submission and Figure 3.5 compares our submissions with the top 7 teams.

Submission	Data size	Accuracy	Ranking
	High	93.89	7
#1	Medium	67.43	8
	Low	3.74	22
	High	**94.66**	**3**
#2	Medium	67.26	10
	Low	2.43	25
	High	93.97	6
#3	Medium	67.36	9
	Low	3.63	23

Table 3.6: The mean accuracy of our Task1 submissions.

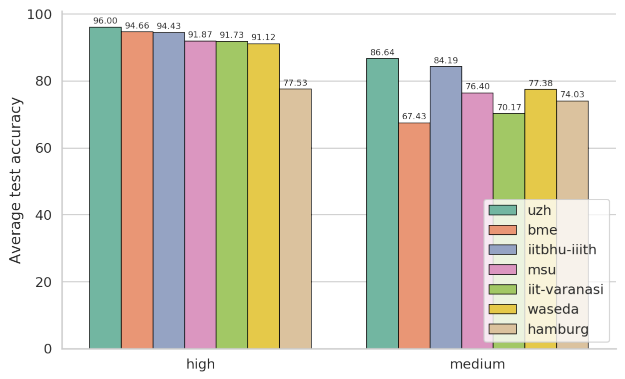

Figure 3.5: SIGMORPHON 2018 Task 1 results of the top 7 teams. Our team is depicted in orange.

3.2.3 Task2: Inflection in Context

In this section we describe our system for Task2 Track1, then explain how the model for Track2 differs from the model for Track1. The development datasets for Task2 have two versions: covered and uncovered. An example is provided in Section 3.2.1.2. Figure 3.6 illustrates the model at a single timestep (decoding one character). The model has several inputs (colored purple):

target lemma The lemma of the target word. The inflected form of this lemma is the expected output.

left/right token context The other (inflected) tokens in the sentence. Left context refers to the tokens preceding the covered token and right context refers to the ones succeeding it.

left/right lemma context The lemmas of the preceding and succeeding tokens.

left/right tag context The corresponding tags of the preceding and succeeding tokens.

previously decoded symbol Start-of-sequence at the first timestep, then the last symbol produced by greedy decoding.

The left and right contexts are encoded separately in the following way. Each token and lemma are encoded by a bidirectional character LSTM, preceded by a character embedding, and the tag sequence

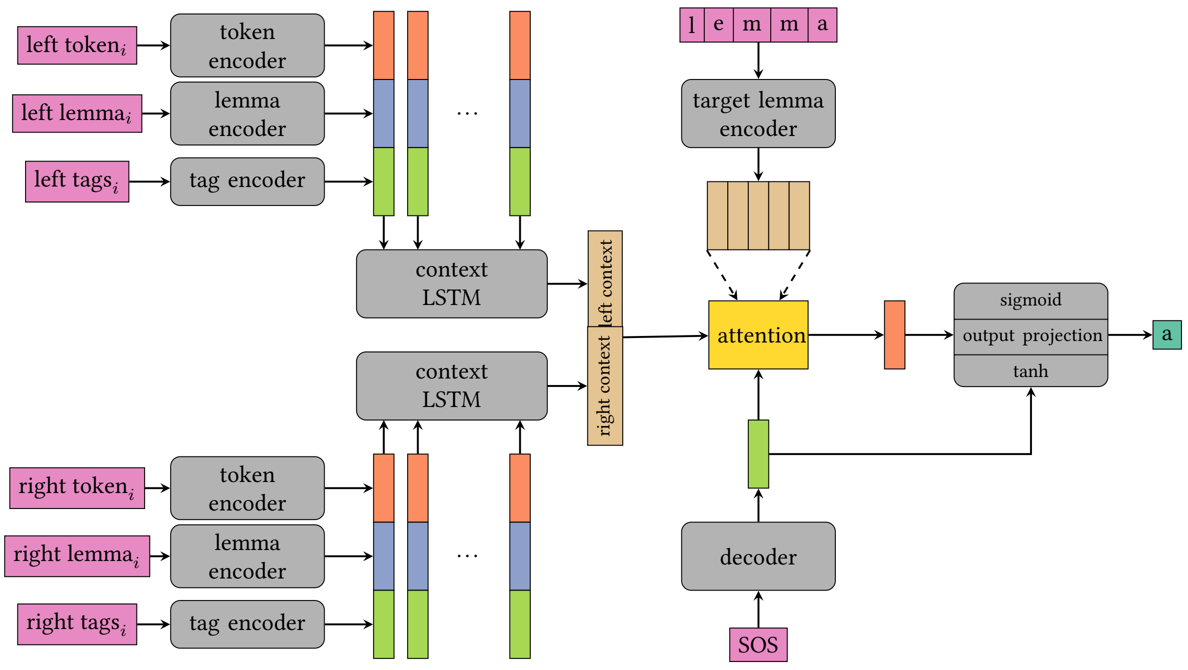

Figure 3.6: Task2 architecture. The figure illustrates the first timestep of decoding. The output of this step is fed back to the decoder in the next timestep. The target lemma encoder's hidden state is used to initialize the decoder hidden state (not pictured for the sake of clarity). The same coloring scheme is used as in Figure 3.3.

of the corresponding token are encoded by a separate biLSTM and tag embedding. The lemma and the token share their alphabet and our experiments showed that sharing the encoder results in a slight improvement in accuracy. By taking the last output of each of the three encoders, we acquire three fixed dimensional vector representation for each token. We concatenate these and use another biLSTM (context LSTM) to create a single vector representation of the left/right context. The context LSTM is shared by the left and the right context. The target lemma is encoded by the same encoder as the other lemmas and inflected tokens and the output is used by the attention mechanism. The last hidden state of the encoder is used to initialize the hidden state of the decoder. Decoding is similar to the autoregressive process used in Task1 but there is only one attention mechanism and it attends to the target lemma encoder outputs. Attention weights are computed using the concatenation of the decoder output at a single timestep and the left and right context vectors. The output of the attention module is concatenated with the decoder output, passed through a *tanh* and an output projection and finally a softmax layer outputs a distribution over the character alphabet of the language. Similarly to our Task1 model, the ground truth is fed to the decoder at training time and the greedily decoded character at inference time. The cross entropy of the output distributions and the ground truth is used as a loss function.

Our model for Track2 is very similar to the model for Track1, except the left and right lemma and tag encoders are missing and the context vectors are derived only from the left and right tokens.

3.2.3.1 Experimental setup

Since our experiments for Task2 were significantly slower than the ones for Task1, we were unable to run extensive parameter search. We did perform a smaller version of the same random search using the parameter ranges listed in Table 3.7. We chose the French dataset with medium setting, which is about 10,000 tokens. The average length of one experiment was 100 minutes and we were able to run 38 experiments. We ran the best configuration of the 38 on each language and each data size at least once. Since our parameter search was very limited, we also varied the parameters manually and tried other combinations. The exact configurations are available on the GitHub repository. All experiments were run on NVIDIA GTX TITAN X 12GB GPUs.

Parameter	**Values**
batch size	8, 16, 32, 64
dropout	0.0, 0.2
early stopping window	5, 10
char embedding size	30, 40, 50
context hidden size	64, 128, 256
context num layers	1, 2
decoder num layers	1
tag embedding size	10, 20, 30
tag num layers	1, 2
word hidden size	64, 128, 256
word num layers	1, 2

Table 3.7: Predefined parameter ranges used for Task2 parameter search.

Task2 uses a subset of the parameters that Task1 uses, so we were able to train the ”same” configuration emerged as the best one during the limited hyperparameter search. We also tried using 2 layers instead of 1 layer in every encoder and decoder. Unfortunately time constraints did not allow running more experiments.

3.2.3.2 Submission and results

For both Track1 and Track2 we only submitted one system, the output of the highest scoring model on the development dataset. We finished in 2nd place in both tracks (Figure 3.7). Table 3.8 lists our detailed results.

	Track1			**Track2**		
	high	med	low	high	med	low
de	73.21	56.83	30.64	64.61	52.17	27.81
en	76.23	66.77	61.33	69.89	64.05	56.90
es	56.10	42.50	29.17	41.65	32.12	27.77
fi	53.75	22.11	10.29	30.24	17.15	8.89
fr	67.21	51.12	26.27	45.42	23.63	9.57
ru	67.67	38.76	21.59	56.73	33.73	19.68
sv	65.64	41.91	26.06	54.26	34.89	22.34

Table 3.8: Task2 results.

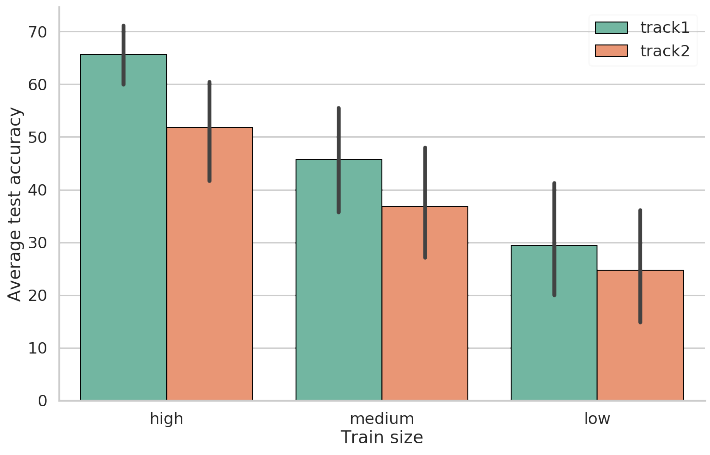

Figure 3.7: Our SIGMORPHON 2018 Task 2 results.

3.2.4 Conclusion

We presented our submissions for the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection. We employed variations of sequence-to-sequence or encoder-decoder networks with Luong attention. Our experiments for Task1 suggest that at the current data size, the model is very sensitive to random initialization, so we used an ensemble of many systems, which placed 2nd of all teams in the high data setting. We also placed 2nd in both tracks of Task2. Our code and configuration files including the random seeds are available on GitHub.

Chapter 4

Neural Pattern Matching

Deep neural networks are successful at various morphological tasks as exemplified in the yearly SIGMORPHON Shared Task (Cotterell et al., 2016, 2017, 2018). However these neural networks operate with continuous representations and weights which is in stark contrast with traditional, and hugely successful, rule-based morphology. There have been attempts to add rule-based and discrete elements to these models through various inductive biases (Aharoni and Goldberg, 2016).

In this chapter we tackle two morphological tasks and the copy task as a control with an interpretable model, SoPa. Soft Patterns (Schwartz et al., 2018) or SoPa is a finite-state machine parameterized with a neural network, that learns linear patterns of predefined size. The patterns may contain epsilon transitions and self-loops but otherwise are linear. *Soft* refers to the fact that the patterns are intended to learn abstract representations that may have multiple surface representations, which SoPa can learn in an end-to-end fashion. We call these surface representations *subwords*, while the abstract patterns, *patterns* throughout the paper.

An important upside of SoPa is that interpretable patterns can be extracted from each sample. Schwartz et al. (2018) showed that SoPa is able to retrieve meaningful word-level patterns for sentiment analysis. Each pattern is matched against every possible subword and the highest scoring subword is recovered via a differentiable dynamic program, a variant of the forward algorithm.

We apply this model as the encoder of a *seq2seq* model, and add an LSTM decoder. We initialize the decoder's hidden state with the final scores of each SoPa pattern and we also apply Luong's attention on the intermediate outputs generated by SoPa. We call this model SoPa Seq2seq. We compare each setup to a sequence-to-sequence with a bidirectional LSTM encoder, unidirectional LSTM decoder and Luong's attention.

We show that SoPa Seq2seq is often competitive with the LSTM baseline while also interpretable by design. SoPa Seq2seq is especially good at morphological analysis, often surpassing the LSTM baseline, which confirm our linguistic intuition, namely that subword patterns are useful for extracting morphological information. We also compare these models using a generalized form of Jaccard-similarity and we find that some trends coincide with linguistic intuition.

This work (Ács and Kornai, 2020) was awarded best paper at the Hungarian Computational Linguistics Conference in 2020. Our code is available GitHub[1].

[1]https://github.com/juditacs/deep-morphology

4.1 SoPa: A Weighted Finite-State Automaton RNN

SoPa is a collection of a predefined number of patterns that score each possible subword and the highest score is used in the final representation of a sequence. We first describe a single pattern. A pattern is a WFSA-ε, but hard constraints are imposed on its shape, and its transition weights are given by differentiable functions that have the power to capture concrete words, wildcards, and everything in between. The model is designed to behave similarly to flexible hard patterns of traditional WFSAs, but to be learnable directly and "end-to-end" through backpropagation. Importantly, it will still be interpretable as a simple, almost linear-chain, WFSA-ε.

Each pattern has a sequence of d states[2]. Each state i has exactly three possible outgoing transitions: a self-loop, which allows the pattern to consume a word without moving states; a main path transition to state $i+1$, which allows the pattern to consume one token and move forward one state; and an ε-transition to state $i+1$, which allows the pattern to move forward one state without consuming a token. All other transitions are given a score of 0. When processing a sequence of text with a pattern p, it starts with a special `START` state and only moves forward (or stays put) until reaching the special `END` state. A pattern with d states will tend to match token spans of length $d-1$ (but possibly shorter spans due to ε-transitions, or longer spans due to self-loops). See Figure 4.1 for an illustration.

The transition function, T, is a parameterized function that returns a $d \times d$ matrix. For a word x:

$$[\mathbf{T}(x)]_{i,j} = \begin{cases} E(\mathbf{u}_i \cdot \mathbf{v}_x + a_i), & \text{if } j = i \text{ (self-loop)} \\ E(\mathbf{w}_i \cdot v_x + \mathbf{b}_i), & \text{if } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{4.1}$$

where $\mathbf{u}_i$ and $\mathbf{w}_i$ are vectors of parameters, a_i and b_i are scalar parameters, $\mathbf{v}_x$ is a fixed pre-trained word vector[3] for x, and E is an encoding function, typically the identity function or sigmoid.

ε-transitions are also parameterized but do not consume a token and depend only on the current state:

$$[\mathbf{T}(\varepsilon)]_{i,j} = \begin{cases} E(c_i), & \text{if } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{4.2}$$

Tokens vs. wildcards. Traditional hard patterns distinguish between tokens and wildcards. SoPa does not explicitly capture the notion of wildcards, but the transition weight function can be interpreted in those terms. Each transition is a logistic regression over the next embedding vector $\mathbf{v}_x$. For example, for a main path out of state i, T has two parameters, $\mathbf{w}_i$ and b_i. If $\mathbf{w}_i$ has a large magnitude and is close to the embedding vector for some token y (e.g., $\mathbf{w}_i \approx 100v_y$), and b_i is a large negative bias (e.g., $b_i \approx -100$), then the transition is essentially matching the specific token y. Whereas if $\mathbf{w}_i$ has a small magnitude ($\mathbf{w}_i \approx 0$) and b_i is a large positive bias (e.g., $b_i \approx 100$), then the transition is ignoring the current token and matching a wildcard. Similarly to Schwartz et al. (2018), we do not try to retrieve wildcard matches based on activation magnitue but rather extract the actual subword a pattern matches.

[2]Schwartz et al. (2018) use patterns of varying lengths between 2 and 7, we found that patterns between 3 and 5 are better suited for morphology.

[3]Schwartz et al. (2018) used GloVe vectors to initialize the word embeddings. We use randomly initialized character embeddings instead.

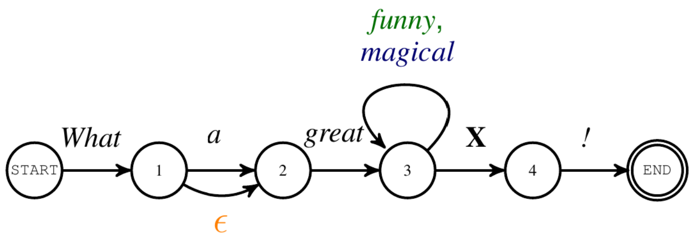

Figure 4.1: A representation of a surface pattern as a six-state automaton. Self-loops allow for repeat- edly inserting words (e.g., "funny"). ε-transitions allow for dropping words (e.g., "a"). Source: (Schwartz et al., 2018).

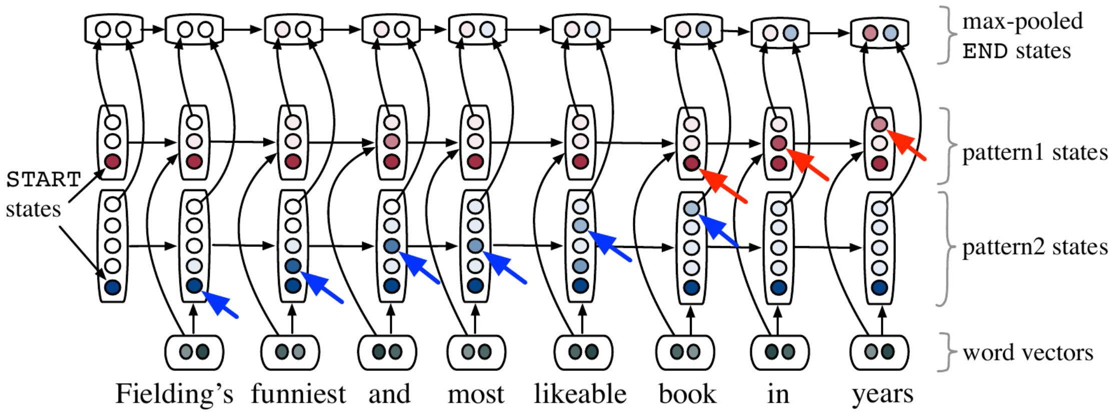

Figure 4.2: State activations of two patterns as they score a document. *pattern1* (length three) matches on "in years". *pattern2* (length five) matches on "funniest and most likeable book", using a self-loop to consume the token "most". Active states in the best match are marked with arrow cursors. Source: (Schwartz et al., 2018).

Scoring sequences. Having described how a single pattern matches a span exactly while consuming the whole span, we now explain how these matches are aggregated over all matches on subspans of the input sequence. In this sense, SoPa is more similar to regular expression *search* than *match*. Depending on the choice of semiring, either the Forward or the Viterbi algorithm can be used to find the highest-scoring match. The exact formlation is available in (Schwartz et al., 2018). We opted for the Viterbi algorithm (max-product semiring). Keeping backpointers during the Viterbi forward pass allows retrieving the matching subspans which we rely on in Section 4.5. Finally, the pattern scores are stacked into one vector and passed through an MLP. We use this vector to initialize the decoder of SoPa Seq2seq. We also use the intermediary pattern scores as inputs for the attention mechanism.

Language	Family	Genus	sample	lemma	paradigm	alphabet	F/L	POS
Arabic	Afro-Asiatic	Semitic	138k	4007	196	45	26.3	NVA
Turkish	Altaic	Turkic	213k	3017	186	46	54.7	NVA
Quechua	Hokan	Yuman	178k	1003	553	22	146.8	NVA
Albanian	Indo-European	Albanian	14k	587	59	27	17.4	NV
Armenian	Indo-European	Armenian	259k	6991	134	46	35.3	NVA
Latvian	Indo-European	Baltic	129k	7238	78	34	10.3	NVA
Lithuanian	Indo-European	Baltic	33k	1391	139	56	20.1	NVA
Irish	Indo-European	Celtic	45k	7299	53	53	3.3	NVA
Danish	Indo-European	Germanic	25k	3190	14	44	7.7	NV
German	Indo-European	Germanic	171k	15032	37	63	4.5	NV
English	Indo-European	Germanic	115k	22765	5	65	4.0	V
Icelandic	Indo-European	Germanic	76k	4774	44	54	10.9	NV
Greek	Indo-European	Greek	147k	11872	118	76	6.5	NVA
Kurdish	Indo-European	Iranian	203k	14143	128	61	14.3	NVA
Asturian	Indo-European	Romance	29k	436	223	32	49.5	NVA
Catalan	Indo-European	Romance	81k	1547	53	35	40.6	V
French	Indo-European	Romance	358k	7528	48	44	35.3	V
Bulgarian	Indo-European	Slavic	54k	2413	95	31	18.9	NVA
Czech	Indo-European	Slavic	109k	5113	147	62	10.0	NVA
Slovenian	Indo-European	Slavic	59k	2533	94	56	8.9	NVA
Georgian	Kartvelian	Kartvelian	74k	3777	109	33	17.5	NVA
Adyghe	NW Caucasian	NW Caucasian	20k	1635	30	40	11.9	NA
Zulu	Niger-Congo	Bantoid	49k	566	249	46	57.2	NVA
Khaling	Sino-Tibetan	Mahakiranti	156k	591	432	32	91.5	V
Estonian	Uralic	Finnic	27k	886	64	26	28.0	NV
Finnish	Uralic	Finnic	1M	57165	97	50	27.1	NVA
Livvi	Uralic	Finnic	63k	15295	104	55	4.0	NVA
Northern Sami	Uralic	Saami	62k	2103	80	31	25.9	NVA
Hungarian	Uralic	Ugric	517k	14883	93	53	34.1	NV

Table 4.1: Dataset statistics. The languages are sorted by language family. F/L refers to the form-per-lemma ratio. POS indicates which part of speech are present in the dataset out of the nouns, verbs and adjectives.

4.2 Data

Universal Morphology or UniMorph is project that aims to improve how NLP handles languages with complex morphology.[4] Specified in Sylak-Glassman (2016), UniMorph has been used to annotate 350 languages from the English edition of Wiktionary[5]. Wiktionary contains inflection tables that list inflected forms of a word. Part of the UniMorph project is converting these tables into *(lemma, inflected form, tags)* triplets such as *(ablak, ablakban, N IN+ESS SG)*. The first tag is the part-of-speech which is limited to the main open classes (nouns, verbs and adjectives) in most languages, `IN+ESS` is the inessive case and `SG` denotes singular.

Our goal is to sample 10000 training, 2000 development and 2000 test examples. We retrieved 109 UniMorph repositories (109 languages) but only 57 languages have at least 14000 samples, the lowest possible number for our purposes. We first prefilter the languages and assign them to languages families and genus using the World Atlas of Languages or WALS[6]. WALS does not contain extinct, constructed or liturgical languages, and we do not incorporate these in our dataset. Out of the 109 languages, 19 have no WALS entry. 29 languages have large enough UniMorph datasets that allow obtaining 10000/2000/2000 samples.[7] Table 4.1 summarizes the dataset.

[4] https://unimorph.github.io/

[5] https://en.wiktionary.org/

[6] https://wals.info/

[7] Albanian has only 1982 test samples but we wanted to include it as a language isolate from the Indo-European family.

Language	Task	Source	Target
Hungarian	morphological analysis	vásároljanak	V SBJV PRS INDF 3 PL
Hungarian	morphological analysis	lepkékben	N IN+ESS PL
English	morphological analysis	hugging	V V.PTCP PRS
French	morphological analysis	désinstalleriez	V COND 2 PL
Hungarian	lemmatization	vásároljanak	vásárol
Hungarian	lemmatization	lepkékben	lepke
English	lemmatization	hugging	hug
French	lemmatization	désinstalleriez	désinstaller
Hungarian	copy	vásároljanak	vásároljanak
Hungarian	copy	lepkékben	lepkékben
English	copy	hugging	hugging
French	copy	désinstalleriez	désinstalleriez

Table 4.2: Dataset examples.

4.3 Tasks

We train both kinds of seq2seq models on three tasks: *morphological analysis*, *lemmatization*, and *copy* or autoencoder. The source sequence is the inflected form of the word in all three tasks, while the target sequence is a list of morphosyntactic tags for morphological analysis, the lemma for lemmatization and the same as the source side for copy. Table 4.2 shows examples for the three tasks.

Inflected words and lemmas are treated as a sequence of characters but tags are treated as standalone symbols. We share the vocabulary and the embedding between the source and target side when training for copy and lemmatization but we use separate vocabularies for morphological analysis.

4.4 Models

We train two kinds of sequence-to-sequence models which only differ in the choice of the encoder. Both models first pass the input through an embedding. We train the embeddings from randomly initialized values and do not use pre-trained embeddings. We use character embeddings with 50 dimensions for character inputs and outputs and tag embeddings with 20 dimensions for morphological tags (only for morphological analysis). The embeddings are shared between the encoder and decoder for lemmatization and copy, since both the source and the target sequences are characters. The output of the source embedding is the input to the encoder module which is a SoPa with 120 patterns in SoPa Seq2seq case and a bidirectional LSTM in the baseline. The decoder later attends on the intermediary outputs of these modules. The final hidden state of the encoder module is used to initialize the decoder. The decoder side of these models is identical in both setups, an LSTM with Luong's attention. All LSTMs have 64 hidden cells and a single layer.

The size of SoPa patterns (3, 4, and 5 in our case) define the number of forward arcs that a pattern has. These may contain epsilon steps and self loops but an epsilon or a self loops is always followed by a main transition (consuming an actual symbol). This means that a 3 long pattern may contain one epsilon and one main transition, two epsilons or two main transitions. Any main transition may be preceded by a self loop. The pattern size includes the start state and the end state. In our experiments we used 3, 4, and 5 long patterns, 40 patterns of each length.

Most of the training details are also identical. We train with batch size 64, and we use early stopping if the development loss and accuracy stop improving for 5 epochs. We maximize the number of epochs in 200 but this is never reached. We save the best model based on development accuracy. We use the Adam optimizer with 0.001 learning rate for all experiments.

SoPa is more difficult to train than LSTMs, so we decay the learning rate by 0.5 if the development loss does not decrease for 4 epochs.

4.5 Model similarity

We define a similarity metric between two SoPa Seq2seq models measured on datasets that share their source side. The target side may differ. The three tasks introduced in Section 4.3, all take inflected word forms as their source sequence, which allows computing our similarity metric between each pair of tasks.

SoPa works with a predefined number of patterns and tries matching each pattern on any subword of the input with a particular length. The highest scoring subword is used in the final source representation. We take the highest scoring $T = 10$ patterns for each input and compare the subwords that resulted in these scores. The metric is defined as the average similarity over the dataset D:

$$\mathrm{Sim}(M_1, M_2, D) = \frac{1}{|D|} \sum_{d \in D} S(M_1(d), M_2(d)), \tag{4.3}$$

where M_1 and M_2 are the models, and S is the similarity of the two representations generated by the encoder side of the models on sample d, defined as:

$$S(M_1(d), M_2(d)) = \frac{1}{2T}\Big(\sum_{p_i \in P_1} \max_{p_j \in P_2} J(p_i, p_j) + \sum_{p_j \in P_2} \max_{p_i \in P_1} J(p_i, p_j)\Big), \tag{4.4}$$

where T is a predefined number of highest scoring patterns on that sample (10 in our experiments), P_1 is the set of T highest scoring patterns of M_1, P_2 is the set of T highest scoring patterns of M_2 and J is the Jaccard similarity of two subwords defined as the proportion of overlapping symbols by the union of all symbols. Jaccard similarity is 0 if there is no overlap and is 1 when the subwords are the same. For each sample, we first choose the highest scoring T patterns from each model, we denote these sets of patterns as P_1 and P_2. Then we find the subwords corresponding to these patterns. We compute the pairwise Jaccard similarities between every element of P_1 and P_2. Then for each pattern, we find the most similar pattern from the other model. The average of these scores is the similarity of the two models on that sample (see Equation 4.4) and the average over all samples (see Equation 4.3) is the similarity of two models on dataset D. This metric is symmetric and it ranges from 0 to 1. Table 4.3 shows a small example of this similarity on the word *ablakban.*

4.6 Results and analysis

We first show that SoPa Seq2seq is competitive with the LSTM Seq2seq baseline, especially for morphological analysis. An output is considered accurate if it fully matches the reference and we do not consider partial matching. Some languages prove to be too difficult for the models, which may be due to the lack of context that is often needed for morphological analysis and orthographic changes often present in lemmatization. We continue our analysis on languages where each of the three tasks are

	$\underline{\text{\^{}ab}}\text{lakban\$}$	$\text{\^{}abl}\underline{\text{akb}}\text{an\$}$	$\text{\^{}ablak}\underline{\text{ban}}\text{\$}$	$\text{\^{}ab}\underline{\text{lakb}}\text{an}$	Max
$\text{\^{}ablak}\underline{\text{ban}}\text{\$}$	0	0.2	1	0.75	1
$\text{\^{}abla}\underline{\text{kba}}\text{n\$}$	0	0.5	0.5	0.75	0.75
$\text{\^{}ab}\underline{\text{lak}}\text{ban\$}$	0	0.5	0	0.167	0.5
$\text{\^{}abla}\underline{\text{kban}}\text{\$}$	0	0.75	0.167	0.333	0.75
Max	0	0.75	1	0.75	J=0.6875

Table 4.3: Similarity (Equation 4.4) between two models M_1 and M_2 on one sample using the 4 highest scoring subwords ($T = 4$) with the subwords underlined. Rows correspond to the highest scoring subwords from M_1 (ban, kba, lak, kban), while columns correspond to the subwords from M_2 (^ab, akb, ban, lakb). A Jaccard similarity matrix (with position information) is constructed. The final similarity is the mean maximum of every row and every column of the matrix.

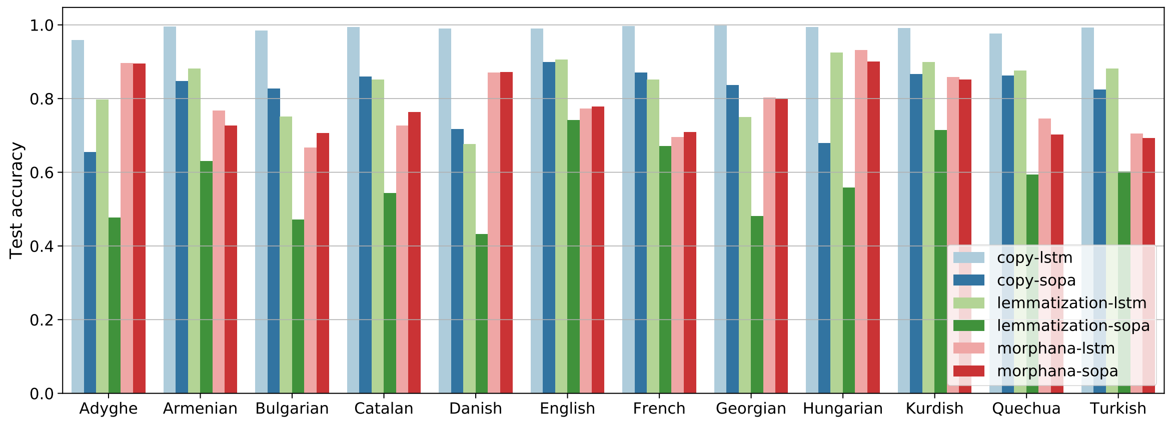

Figure 4.3: Accuracy of SoPa Seq2seq models on each language and task.

performed by SoPa 'reasonably well', which we set to 40% accuracy or higher on the development set. This leaves us with 12 languages. The reason we set a lower limit to accuracy is that we have no reason to believe that a bad model's representation is useful for the task. Figure 4.3 shows the test accuracy in these languages. Lemmatization is consistently the most difficult task for SoPa, while SoPa is on pair with LSTM Seq2seq in morphological analysis, sometimes outperforming it. We attribute this result to the fact that a morphological tag often corresponds to a single morpheme, usually with a few possible surface realizations that SoPa's 'soft' patterns can pick up on. On the other hand lemmatization and copy require regenerating much of the input which is more difficult from an inherently summarized representation such as the one SoPa generates.

We continue by computing the pairwise similarity value defined in Equation 4.3 between the three tasks. Higher values indicate that SoPa finds similar patterns valuable for generating the output. Figure 4.4 shows the pairwise similarity of models trained for the three tasks. We only compute these similarities on samples where the output of *both* models are correct (generally 40-60% of the test samples).

Lemmatization and morphological analysis are the least similar in almost every language. This is not surprising considering that lemmatization is the task of discarding information that morphological analysis needs to correctly tag. Quechua is the only exception from this trend which could be explained by the very rich inflectional morphology (especially at the type-level) that results in lemmas being significantly shorter than inflected forms. This means that copy needs to memorize a lot

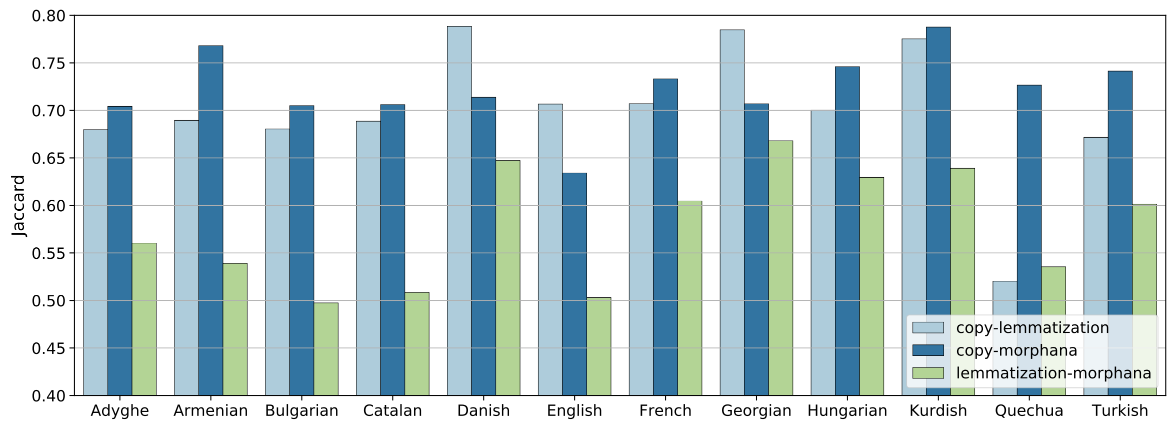

Figure 4.4: Model similarity between all task pairs by language. Higher similarity indicates that two models handle the same source in a more similar way.

language	task	subwords
English	copy	ed,e$,ed$,es,in,at,re,s$,te,ri
English	lemmatization	at,g$,er,in,ng,iz,s$,en,ize,es
English	morphological_analysis	d$,s$,e$,es$,$,ed,ed$,o,ng,g$
French	copy	s$,ss,is,as,ie,ai,z$,nt,ns,en
French	lemmatization	er,s$,t$,nt,ie,ns,ra,is,ri,^d
French	morphological_analysis	s$,t$,z$,nt$,ez$,e$,ai,er,ns$,es$
Hungarian	copy	l$,n$,k$,sz,t$,nk$,kk,el,ok,na
Hungarian	lemmatization	sz,t$,k$,l$,ta,tá,^k,n$,kb,ró
Hungarian	morphological_analysis	l$,t$,n$,k$,ek,a$,$,g$,á$,ak$

Table 4.4: Top subwords extracted from English, French and Hungarian. ^and $ denote word start and end respectively.

more of the source word than lemmatization.

Another trend we observe, is that copy and morphological analysis are more similar than copy and lemmatization in languages with rich inflectional morphology such as Armenian, Hungarian, Kurdish and Turkish and the opposite is true in fusional and morphologically poor languages such as Danish and English. Georgian seems to be an exception.

Finally we demonstrate SoPa's interpretability by extracting the most frequently matched subwords in each language and task. Table 4.4 lists the most common subwords in English, French and Hungarian in each task. It should be noted that these subwords are very short because we used 3, 4 and 5 long patterns that match 2, 3 and 4 characters not including self loops and short patterns simply occur more frequently.

4.7 Conclusion

We presented an application of Soft Patterns – a finite state automaton parameterized by a neural network – as the encoder of a sequence-to-sequence model. We show that it is competitive with the popular LSTM encoder on character-level copy and morphological tagging, while providing interpretable patterns.

We analyzed the behavior of SoPa encoders on morphological analysis, lemmatization and copy by computing the average Jaccard similarity between the patterns extracted from the source side. We found two trends that coincide with linguistic intuition. One is that lemmatization and morphological analysis require patterns that match less similar subwords than the other two task pairs. The other one is that copy and morphological analysis are more similar in languages with rich inflectional morphology.

Part II

Evaluating Pre-trained Language Models

Chapter 5

Background

5.1 History of meaning representation

Word representation models based on word distribution date back to Zellig Harris' seminal 1954 book, *Distributional Structure* (Harris, 1954) which was the first to mention *Bag-of-words* or *BOW* models. BOW represents a passage, usually a sentence, as an unordered (bag) collection of words. The collection may include word counts. Since the order of word is not preserved, most syntactic information is lost.[1].

Both BOW and keyword-based models such as Term Frequency-Inverse Document Frequency use sparse vectors, where each dimension corresponds to one word in the dictionary, as the numeric representation of words which carries significant difficulties for computational models. They also fail to capture any relationship between words and their context.

5.1.1 Word embeddings

A breakthrough came in 2013 when Mikolov et al. (2013) introduced *Word2Vec*, a model that generated dense word embeddings based on a word's co-occurrence patterns in a large corpus. Word2Vec used two architectures: *Continuous Bag-of-Words (CBOW)*, which predicted a word from its context, and *Skip-Gram*, which predicted the context given a word. By training on these tasks, Word2Vec placed semantically similar words close to each other in the embedding space. For example, *king* and *queen* would appear near each other, and algebraic relationships like $\text{king} - \text{man} + \text{woman} \approx \text{queen}$ demonstrated its ability to capture linguistic structure. Mikolov et al. (2013) published the trained Word2Vec embeddings providing the first version of pre-trained embeddings. These embeddings were widely successful at a large variety of NLP tasks and they still remain a fast and lightweight solution for meaning representation in small models.

A similar set of word embeddings named *Global Vectors* or *GloVe* were released the next year (Pennington et al., 2014). GloVe incorporated global co-occurrence statistics into the embedding space. This approach resulted in embeddings that better represented semantic relationships and improved performance across various NLP tasks.

Formally an embedding is defined as $\mathbf{E} \in \mathbb{R}^{|V| \times d}$, where $\mathbf{E}$ is the embedding matrix, V is the vocabulary, and d is the dimension of the embedding. V is usually determined based on the data available, and missing symbols are mapped to a predefined *unknown* symbol. d is usually set to less than 1,000, typically in the 300–600 range.

[1]We experiment with BOW as a perturbation in Section 8.1

Once an embedding is trained on raw text, it is most often used as the first block of a task-specific neural network. The embedding weights may or may not be updated along with the rest of the network parameters. The two approaches are rudimentary versions of *feature extraction* and *fine-tuning*, two use cases we will explore in more detail.

Smaller embeddings, particularly character embeddings, are often trained from scratch as part of end-to-end training for a specific task. We employ this strategy throughout Chapters 3 and Chapter 4.

Despite their success, Word2Vec and GloVe had a major limitation: they produced static embeddings, where a word had a single fixed vector regardless of its meaning in different contexts. For example, the word "bank" would have the same representation whether referring to a financial institution or a riverbank. This lack of contextual awareness hindered their effectiveness in tasks requiring nuanced understanding of language. Multi-sense embeddings extend this by allowing a small set of distinct word senses with different vectors for each sense. For a detailed survey on multi-sense embeddings see (Camacho-Collados and Pilehvar, 2018; Borbély et al., 2016).

5.1.2 Pre-trained contextual embeddings

The next generation of meaning representations use context in a dynamic way. The language model is now a function of the sentence or a longer sequence instead of a single word type. The resulting representation is one or more tensor for every token in the sentence that depend on the context as well. They also often include a pooled representation for the full sentence or a sentence pair suitable for sentence level tasks such as sentiment analysis and sentence pair tasks such as natural language inference.

Perhaps the earliest example of *contextualized language models* was *Contextualized Word Vectors* or CoVe (McCann et al., 2017). CoVe uses the encoder of a trained neural machine translation model as a language model of the source language of the translation. In downstream applications CoVe vectors are concatenated with the 'standard' embedding vectors such as GloVe. McCann et al. (2017) find that CoVe improves the downstream performance and that the quantity of the training data used for training the machine translation model positively correlates with this performance.

The next generation of contextualized word vectors completely replaces GloVe and other pre-trained static embeddings. ELMo or *Embeddings from Language Models* (Peters et al., 2018) is a two-layer biLSTM stacked over a static embedding trained with a language modeling objective on large corpora. The objective is to predict the next or the previous word given its context. This enabled ELMo to produce different embeddings for the same word depending on its usage, significantly improving performance on tasks like named entity recognition and sentiment analysis. Furthermore they suggested a novel approach for task-specific fine-tuning. The output of the embedding layer and the two LSTM layers are linearly combined and used as an input for downstream tasks. The layer weights are task-specific and trained on the particular task's training data. ELMo improved the state of the art (SOTA) results for various English tasks. There are various editions of pre-trained ELMo vectors for dozens of languages.

Arguably the most important breakthrough was the introduction of the BERT model from Google (Devlin et al., 2019). *BERT* or *Bidirectional Encoder Representations from Transformers*, a Transformer-based model, quickly became the gold standard for pre-trained models and it paved the way for a large bulk of NLP research in the following years. The first BERT implementation used TensorFlow, Google's own framework which by then was less popular for research than its competitor, PyTorch. Hugging Face, a machine learning company based in New York City, published the first PyTorch version of BERT along with a model repository open for the public. Within a year the repository had thousands of model checkpoints including a large variety of language and domain-specific BERT model checkpoints. Since the BERT model family is the main focus of Part II. of this thesis, we introduce BERT in more detail in Section 5.2.

5.1.3 Terminology

There are various names used in reference to the new generation of language models, and BERT models in particular. *Contextual* or *contextualized language models (CLMs)* refer to any model that uses the context of a word (and the word itself) to generate the numeric representation of the word. *Pre-trained language models (PLMs)* refer to models with extensive pre-training on unlabeled raw text. *Masked language models (MLMs)* specifically refer to the type of training objective where tokens are masked in the middle of the sentence and the model is tasked to recover them. Throughout this thesis most of the models we analyze are MLMs, but many of our methods are not limited to such models, rather we are interested in the effect of pre-training, therefore we opt to use the term pre-trained language models or PLMs. *Large language models* or LLMs refer to the high number of parameters they have typically at least ten billion and most of these models are generative models while we mainly study encoder models in this work. Most of the models we study are smaller than this and Ács et al. (2023) only covers the BERT family with models at most a couple hundred million parameters.

We should also clarify the difference between models, model families and instances of models. A *model* is an architecture such as a multilayer Transformer. The training objective is usually treated as a part of the model, even though strictly speaking the same model architecture can be trained via different objectives. For example the RoBERTa family is architecturally identical to the BERT family but it does not use the next sentence prediction objective.

A *model family* is a loose term for a group of models that may have some difference in their architecture and their training process, but they share many important traits. The BERT family is a collection of self attention *encoder* models, trained using a masked language modeling objective. Unlike other model families such as the BART family, BERT models are not generative and therefore they have limited success in generative tasks.

A *model instance* or a *pre-trained checkpoint* is a set of weights, a pre-defined vocabulary and a tokenizer. These three define a usable instance of a model. The vocabulary and the tokenizer may be shared between multiple models such as in the case of mBERT and DistilmBERT. Throughout this thesis, the model names refer to pre-trained checkpoints. We also include the models' string id in the Hugging Face model repository.

5.2 BERT

The next set of improvements came from replacing the LSTM with Transformers and by a vast increase in both model size and training data. Google researchers introduces BERT (Devlin et al., 2019) in 2018 pioneering Transformer-based pre-trained language models. It is safe to say that BERT revolutionized the field and it quickly spurred a large number of similar models. The subfield of NLP related to BERT models and their study is called *BERTology*.

BERT or *Bidirectional Encoder Representations from Transformers* is a multilayer Transformer model trained with two objective functions, masked language modeling (MLM) objective and next sentence prediction.

Masked language modeling is the task of predicting words replaced by a dedicated mask symbol. Unlike the traditional language modeling objective MLM allows bidirectional training, in other words, the distribution of the predicted word is a function of both the left and the right context. In practice, the training data generator picks a random 15% of the tokens to be predicted. 80% of those is replaced with a mask symbol, 10% is replaced with a random token and 10% are left unchanged.

Next sentence prediction is inspired by question answering and natural language inference, tasks that require understanding the relationship between multiple sentences instead of modeling a single sentence. Pairs of sentences are extracted from the training data. Half of the examples are actual consecutive sentences, half are not. However one of the immediate successors of BERT, RoBERTa (Liu et al., 2019b) suggested that this may not be necessary as long as there is sufficient training data.

BERT uses the WordPiece tokenizer (Schuster and Nakajima, 2012) and each BERT model uses its own subword vocabulary. BERT is originally designed for sentences or sentence pairs. Sentences are prefixed with a `[CLS]` symbol and suffixed with a `[SEP]` symbol. The two sentences of sentence pairs are separated with `[SEP]` as well. `[CLS]` can be used as a representation for the full sentence or sentence pair. After the input is split into subwords, each subword is mapped to a vector the same way as in word embeddings. A segment embedding that differentiates the first and the second sentence of a sentence pair, is then added to the embedded vectors. Since Transformers are directionless, a position embedding is used to mark word order. This process is depicted in Figure 5.1.

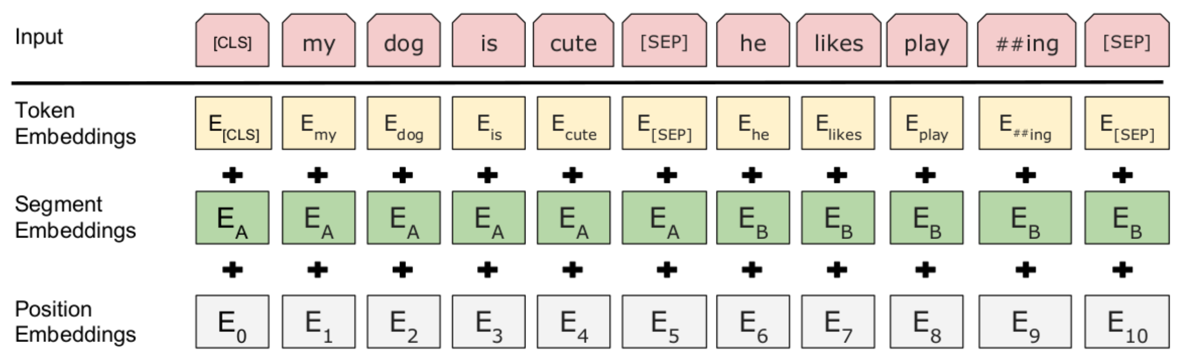
Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 5.1: BERT tokenization. Source Devlin et al. (2019)

Devlin et al. (2019) introduced two ‘sizes’, a base model and a large model. BERT-base has 12 Transformer (Vaswani et al., 2017) layers with 12 attention heads. The hidden size of each layer is 768. BERT-large has 24 layers with 16 heads and 1024 hidden units. BERT-base has 86M parameters without the embedding, BERT-large has 303M parameters without the embeddings. The size of the embedding depends on the size of the vocabulary which is specific to each pre-trained BERT model. BERT-base has 110M parameters, BERT-large has 340M parameters. The original paper introduced 6 checkpoints: cased and uncased English checkpoints for both sizes and a cased and an uncased 100 language model. The latter was quickly dismissed in favor of the cased one. This has been extended with smaller versions since then.

BERT is generally used as the largest part of a downstream model with only a minimal adaptation layer required for classification and other tasks. It is initialized with a pre-trained BERT checkpoint. The full downstream model is then trained further on a task-specific dataset. This approached is called *fine-tuning*. Fine-tuning, illustrated in Figure 5.2, is generally much faster than pre-training, often requiring just one or a few training steps. The end result is a variety of specific BERT instances (Figure 5.3). Hugging Face hosts a free repository for both pre-trained and fine-tuned checkpoints.[2]

[2]https://huggingface.co/models

The other approach, called *feature extraction*, is to fix BERT's weights and only update the adaptation layer or layers. Although feature extraction is generally inferior to fine-tuning, its computational requirements are much more modest. In this thesis we use feature extraction unless noted otherwise.

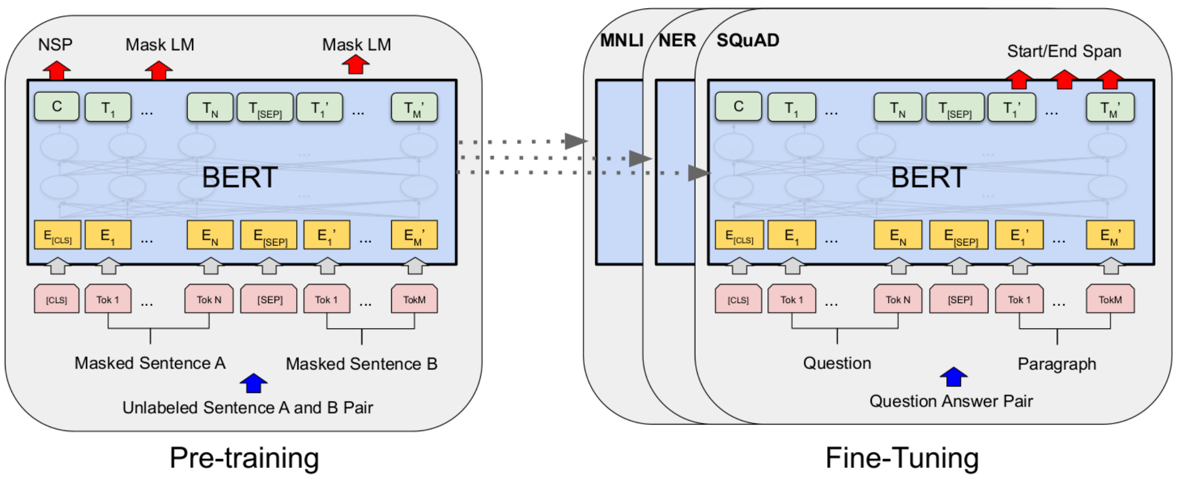

Figure 5.2: BERT fine-tuning for various downstream tasks. Source: Devlin et al. (2019).

5.2.1 Multilingual models

Multilingual models are pre-trained on corpora in multiple languages, usually without any distinction as to which language a particular sentence is in. They generally use a single vocabulary and tokenizer for all supported languages. Some multilingual models are limited to a few languages such as SlavicBERT (Arkhipov et al., 2019), which supports Russian, Bulgarian, Czech and Polish, and massively multilingual models that support over 100 languages. The most prominent examples of massively multilingual models are mBERT (Devlin et al., 2019) and XLM-RoBERTa (Conneau and Lample, 2019). Both models were trained on over 100 languages including Hungarian.

5.2.2 Multilingual BERT

Multilingual BERT (mBERT) was released along with BERT, supporting 104 languages. The main difference is that mBERT is trained on text from many languages. In particular, it was trained in a resource-balanced manner. Languages with many Wikipedia articles were undersampled while the low-resource languages oversampled, using Wikipedia dumps with a shared vocabulary across the supported languages. As a BERT-base model, its 12 Transformer layers have 86M parameters, while its large vocabulary BERT-base model has a much smaller vocabulary and therefore a smaller embedding with 23M parameters. The combined parameter count of the English BERT, 110M parameters, is mistakenly listed as the parameter count of mBERT on the website[3].

5.2.3 XLM-RoBERTa

XLM-RoBERTa is a hybrid model mixing together features of two popular Transformer-based models, XLM (Lample and Conneau, 2019) and RoBERTa (Liu et al., 2019b). XLM is trained on both masked

[3]https://github.com/google-research/bert/blob/master/multilingual.md

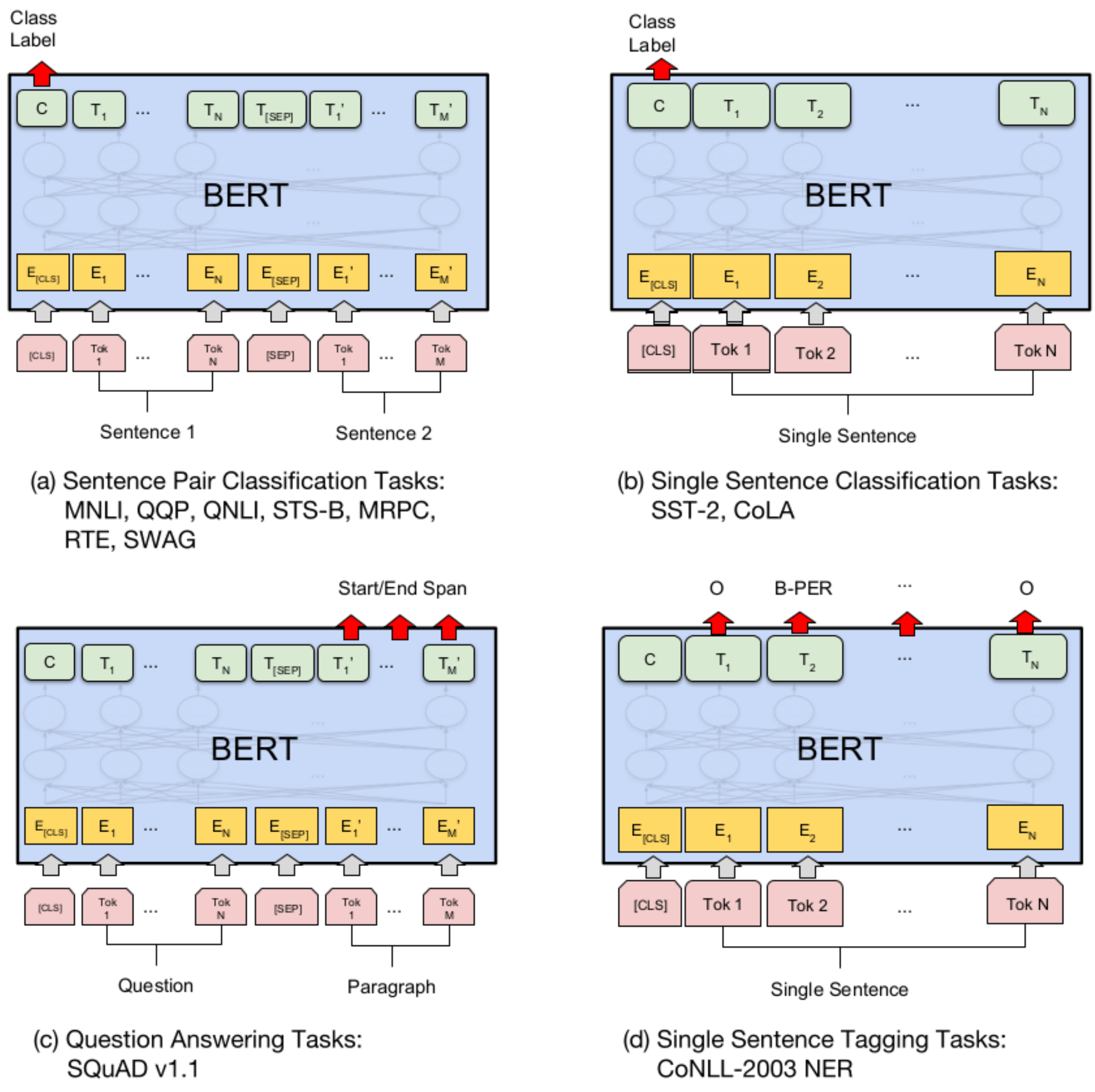

Figure 5.3: BERT usage for sequence classification and sequence tagging. Source: Devlin et al. (2019).

language modeling (MLM) and Translation Language Modeling (TLM) objective on parallel sentences. In contrast, XLM-RoBERTa is trained using the MLM objective only, like RoBERTa. The main difference between XLM-RoBERTa and RoBERTa remains the scale of the corpora they were trained on: XLM-RoBERTa's training corpora counts five times more tokens and more than twice as many (278M with embeddings) parameters than RoBERTa's 124M (Conneau et al., 2020). Another major difference between these two models is that XLM-RoBERTa is trained in self-supervised manner, while the parallel corpora for XLM is a supervised teaching signal. In the Cross-lingual Natural Language Interface (XNLI; Conneau et al., 2018b) evaluation of mBERT, XLM, and XLM-RoBERTa, the latter outperformed the other CLMs in all the languages in XNLI. We abbreviate XLM-RoBERTa as *XLM-R* in the figures and tables for brevity.

5.2.4 Other multilingual models

As of 2022, there were a few other massively multilingual general domain models available in the model repository hosted by Hugging Face[4]:

XLM-large is the large version of XLM-RoBERTa. It uses the BERT-large architecture but otherwise its setup is identical to XLM-RoBERTa. Its string id is `xlm-roberta-large`.

XLM-MLM-100 is also a larger variant of XLM-RoBERTa with 16 layers instead of 12. Its string id is `xlm-mlm-100-1280`.

distilbert-base-multilingual-cased is a *distilled* version of mBERT. It cuts the parameter budget and inference time by roughly 40% while retaining 97% of the tutor model's NLU capabilities. Its string id is `distilbert-base-multilingual-cased`.

mT5 (Xue et al., 2021) is a massively multilingual *generative* model. To the best of our knowledge, this is the only general purpose generative model that supports Hungarian and a large portion of the languages we study.

We include all of the above models in our main analysis (cf. Section 6.5) but our main focus is on mBERT and XLM-RoBERTa, the two mid sized models that are comparable to each other and arguably the most popular choices for encoder oriented tasks. This list has grown since 2022 but the focus has shifted to generative models and our study is better suited for encoder models.

5.2.5 Monolingual models

Monolingual model checkpoints are trained on a single language. As with all BERT models, their subword vocabulary is trained before any of the model weights. These vocabularies usually include the ASCII character set and various subwords comprising of Latin or pure ASCII characters even if the language itself uses a different alphabet. Monolingual models usually use a 5 to 10 times smaller vocabulary than massively multilingual ones. In this thesis we use 5 monolingual BERT checkpoints: EngBERT, HuBERT, RuBERT (Russian), FinBERT (Finnish) and EstBERT (Estonian).

5.3 Subword tokenization

Language models model natural language as a sequence of symbols. The choice of symbol determines the vocabulary size and the tokenization algorithm. The most common choices are:

Characters. Defined as Unicode code points, are readily available without any sophisticated tokenization. On the other hand, they seldom convey meaning on their own, fully shifting the task of semantic modeling to other modeling components.

Morphemes. Morphemes are the smallest linguistic units with meaning. In practice, morphemes are rarely used as symbols for several reasons. The main one is that recovering morpheme boundaries is non-trivial. Even in languages with concatenative morphology, assimilation or sandhi effects are common, which makes it hard to segment morphemes at character boundaries. There have been various attempts at using morpheme boundaries for tokenization (Cotterell and Schütze, 2015; Cao and Rei, 2016).

[4]https://huggingface.co/models

Subwords. Subwords are character spans shorter than words but, preferably longer than one character. The models we evaluate in Part II., use subword tokenization so we provide a more detailed introduction in Section 5.3.

Words. Word tokenization was the *standard* way of modeling language for English. English has a very limited morphology and the ratio of inflected forms to lemmas is low.[5] However, this approach is insufficient for morphologically richer languages.

Ever since the introduction of contextual models, language models almost exclusively use subword tokenization. Subword tokenization proved to be a simple yet effective way of tokenization even when multilingual models use a shared vocabulary across widely different languages (Ács, 2019).

Subwords are character spans shorter than words but, preferably longer than one character. They are usually acquired through simple character frequency-based heuristics such byte pair encoding. Subword vocabularies are built off-line and they are independent of the model weights. Indeed, different models often share their subword vocabularies.

Subword vocabularies are derived from simple frequency based methods such as byte pair encoding (Gage, 1994; Sennrich et al., 2016) and its extension, the *SentencePiece* (Kudo, 2018) algorithm. Unlike the model weights, which are all trained simultaneously (end-to-end training), these vocabularies are built offline, before any of the model weights are initialized. The model's embedding is initialized to match the size of the subword vocabulary.

Once a subword vocabulary is built, the segmentation of a word can be recovered via greedy decoding. Subword tokenization has multiple advantages compared to word tokenization. One is that we directly control the vocabulary size when we build the vocabulary. Another one is that out-of-vocabulary symbols can be reduced to a minimum if we match the alphabet of the language since subword tokenizers fall back to character tokenization when necessary. The disadvantage is that although subwords are longer than one character, unlike morphemes, they are not necessarily meaningful. All language models that we evaluate in this thesis use subword tokenization but we manipulate the tokenizer in some cases.

5.3.1 Multilingual subword tokenization

Multilingual models use a shared vocabulary across all languages. These shared vocabularies are trained on mixed language data often without any language specific label. This results in overlapping segments for different languages. These overlaps are sometimes desirable such as sharing subwords for named entities or related words. An example is *Hollandia*, the Netherlands in Hungarian, segmented as `Holland + ia` by mBERT, where the subword Holland is meaningful in both English and Hungarian and their meaning is practically the same. On the other hand there are many false friends. One of the most common subwords in both English and Hungarian is *most* which means 'now' in Hungarian.

5.3.2 Cased and uncased models

Many of these models have *cased* and *uncased* versions. Uncased versions were trained on lowercased data with the diacritic marks removed. The uncased models, though lossy, are suitable for postprocess-

[5]The UNIMORPH dataset, which is derived from inflection tables in Wiktionary, only defines a total of 5 distinct inflection paradigms for English.

Model	Language	Input token	Tokenization	Restored token
mBERT uncased	Hungarian	Nyíregyháza	ny + ire + gy + haza	nyiregyhaza
mBERT cased	Hungarian	Nyíregyháza	Ny + ír eg y háza	Nyíregyháza
English BERT uncased	Hungarian	Nyíregyháza	ny ire gy ha za	nyiregyhaza
English BERT cased	Hungarian	Nyíregyháza	N + y + í + re + gy + h + á + za	Nyíregyháza

Table 5.1: Examples of cased and uncased tokenization.

ing tasks such as adding punctuation marks to the output of an automatic speech recognizer (Nagy et al., 2021).

Uncased models often perform aggressive 'normalization' by removing not only diacritics - a practice often used in informal writing in Hungarian - but also removing important markers such as *dakuten* in Japanese. Dakuten are part of the Japanese *kana* system, which consists of multiple syllabaries, of which *hiragana* and *katakana* are the most prominent in modern Japanese. Dakuten are most often used to indicate when a consonant of a syllable should be voiced. Similarly Arabic short vowel markers are removed. Table 5.1 lists some examples of cased and uncased tokenization. English models perform similar case removal and they are also missing many but not all non-English characters as shown in the uncased examples in Table 5.1. We mostly focus on cased models in this thesis.

5.3.3 The tokenizers of mBERT and XLM-RoBERTa

Each BERT model has its own vocabulary. The vocabulary is trained before the model, not in an end-to-end fashion like the rest of the model parameters. mBERT and XLM-RoBERTa both share the vocabulary across 100 languages with no distinction between the languages. This means that a subword may be used in multiple languages that share the same script. The subwords are differentiated whether they are word-initial or continuation symbols. mBERT, trained using WordPiece (Schuster and Nakajima, 2012), marks the continuation symbols by prefixing them with ##. In contrast, XLM-RoBERTa, acquired with SentencePiece (Kudo, 2018), marks the word-initial symbols rather then the continuation symbols, with a Unicode lower eights block. The idea is that both these marks are almost non-existent in natural text so it is easy to recover the original token boundaries.

mBERT uses a vocabulary with 118k subwords, while XLM-RoBERTa's vocabulary has 250k subwords. This means that XLM-RoBERTa tends to generate fewer subwords for a given token, because longer partial matches are found more easily. Ács (2019) defines a tokenizer's *fertility* as the proportion of subwords to tokens. The higher this number is, the more often the tokenizer splits a token.

5.4 Evaluating PLMs

PLMs are most often implemented as multilayer Transformer networks with over a hundred million trainable parameters that implement a non-linear function that assigns a continuous representation to the input sentence or sentence pair. The *quality* of this representation is hard to assess due to multiple reasons. First, the function learned by the model is highly non-linear, therefore it is impossible to pinpoint single factors in the input, such as the presence or absence of particular words, responsible for the output. Second, the training objectives are self-supervised, they are generated form the input data via masking or simply next word prediction as in traditional (e.g. hidden Markov) language models. This means that the pre-training process never actually optimizes for a downstream application. This is alleviated by fine-tuning, where we optimize the model on a downstream application but the labeled data used for fine-tuning is usually orders of magnitudes smaller than the unlabeled data used for the pre-training. For these reasons, PLMs are considered *black box* models.

The quantitative evaluation of PLMs can be performed via intrinsic and extrinsic measures. Intrinsic measures include training metrics such as the training and development loss, and depending on the type of the model, language modeling metrics such as perplexity or masked language modeling accuracy. Extrinsic evaluation is performed via surrogate tasks, where the language model is a component of a larger model aimed at solving some downstream task.

There is also a growing body of work aimed at interpreting the inner workings of PLMs. Interpretability is crucial in real world applications where accountability is a requirement (Doshi-Velez and Kim, 2017; Markus et al., 2021; Holzinger et al., 2017; Tan et al., 2023; Zhao et al., 2024).

In this chapter we perform both intrinsic and extrinsic evaluation. Our intrinsic method is probing (6) and our extrinsic evaluation is aimed at two downstream tasks that are available in many languages, POS tagging and NER. We mainly focus on evaluating multilingual and non-English monolingual models.

5.4.1 Intrinsic measures

Intrinsic measures include training statistics such as the loss as a function of the training steps. During training, we use the train set to compute gradients and update the model weights and we use the development set as a held out set for independent evaluation. Losses and other measures may be computed on the development set but they are not used for updating the model weights. The test set is only used to assess the fully trained model. Depending on the objectives used for training a model, multiple types of intrinsic measure are available.

Loss. The loss or cost function is a function with two inputs, the model output and the reference output i.e. the *correct* output. The loss function assigns a real value to these inputs that quantifies the *goodness* of the model output. Lower loss values indicate that the model output is *closer* to the reference. In the case of models trained via backpropagation, we require that these losses are differentiable with respect to the model parameters. In practice, this limits the choice to a few major variants such as cross entropy for classification or mean squared error for regression. Language modeling objectives are formulated as classification tasks, where the model has to predict either the next word (generative models) or a masked word in its context (BERT). The task is to pick the correct word or subword from a predefined vocabulary, a classification task often with more than 100,000 different labels.

Perplexity. Perplexity quantifies how well a probability distribution predicts a given sample. The perplexity of a probability distribution p is defined as

$$PP(p) = \prod_x p(x)^{-p(x)} \tag{5.1}$$

where x ranges over all samples.

Masked language modeling perplexity. Models from the BERT family are trained with a masked language modeling objective, i.e. the model has to predict a subset of the subwords that are masked. MLM perplexity is the perplexity measured on these subwords.

5.4.2 Downstream performance

The most straightforward way of assessing a model's capabilities is via downstream applications. The pre-trained checkpoints for these models are intended for such usage.

Despite the wealth of multilingual and language-specific models, most evaluation methods are limited to English, especially for the early models such as the original 6 BERT checkpoints (cf. Section 5.2) and we only find a handful of papers aimed at multiple languages. Devlin et al. (2019) showed that the original mBERT outperformed existing models on the XNLI dataset (Conneau et al., 2018b), a 16 language translation of the test set of NLI. mBERT was further evaluated by Wu and Dredze (2019) for 5 tasks in 39 languages, which they later expanded to over 54 languages for part-of-speech tagging and dependency parsing and 99 languages for named entity recognition (Wu and Dredze, 2020). All three tasks include Hungarian as one of the low resource languages but they do not analyze the specific languages in detail.

mBERT has been assessed on a variety of multilingual tasks such as dependency parsing (Kondratyuk and Straka, 2019) or constituency parsing Kitaev et al. (2019). (Kondratyuk and Straka, 2019) tested all 75 languages that have UD treebanks including Hungarian. mBERT's multilingual capabilities have been explored for NER, POS and dependency parsing in dozens of language by Wu and Dredze (2019) and Wu and Dredze (2020). The surprisingly effective multilinguality of mBERT was further explored by Dufter and Schütze (2020).

5.4.3 Benchmarks

Benchmarks are collections of one or more annotated dataset with standard train and test splits. These are a popular way of assessing NLP models in a reproducible and comparable manner. Many benchmarks have online submission systems with leaderboards in the style of machine learning competition websites such as Kaggle[6].

Most tasks in these benchmarks are sentence or sentence pair classification tasks with tens of thousands of training samples. The limited training set makes it possible to train the models on modest hardware. Although many of these benchmarks predate BERT, BERT-based solutions quickly took over the state of the art in 2018. Although since then LLMs have taken over, BERT and its successors still offer reasonable performance with modest computational requirements.

5.4.3.1 English benchmarks

As with any other aspect of NLP, English is by far the highest resources language when it comes to standardized benchmarks. These benchmarks consist of a variety of datasets and tasks designed to measure aspects such as linguistic understanding, reasoning, and the ability to handle real-world complexities. Prominent examples include the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018), which assesses models across tasks like sentiment analysis, sentence similarity, and natural language inference. Its successor, SuperGLUE (Wang et al., 2019), introduces more challenging tasks to evaluate advanced reasoning abilities. Similarly, datasets such as SQuAD (Stanford Question Answering Dataset) (Rajpurkar et al., 2016) test reading comprehension, requiring models to extract answers from text passages. By offering standardized evaluation frameworks, these benchmarks provide a means to compare models consistently.

However, English NLP benchmarks are not without limitations. Many benchmarks emphasize specific tasks or linguistic phenomena, potentially overlooking broader, real-world complexities like

[6] https://www.kaggle.com/

multilingual contexts or domain-specific knowledge. For instance, while datasets like GLUE and SuperGLUE focus on general language tasks, they may fail to capture nuances in technical, legal, or medical language. Additionally, models optimized for these benchmarks can exhibit overfitting behaviors, excelling in benchmark tasks without achieving robust generalization. Newer multilingual benchmarks such as XTREME (Hu et al., 2020) and BIG-bench (bench authors, 2023) address this by including a wider range of linguistic and cognitive tasks.

5.4.3.2 Multilingual benchmarks

XTREME (Hu et al., 2020) is a multilingual set of benchmarks that covers 40 languages. It has 4 types of tasks, sentence classification, structured prediction (POS and NER), sentence retrieval and question answering. Each task consists of 2 or 3 different benchmarks, with 9 benchmarks in total. Not all tasks are available in all languages, for example only 4 tasks are available in Hungarian. Since Hungarian is only available in the structured prediction (a.k.a. sequence tagging) tasks and a small machine translation benchmark, we do not use XTREME with this thesis.

XNLI (Conneau et al., 2018b) is a multilingual natural language inference dataset. It is a translation of the development and test splits of the MultiNLI (Gururangan et al., 2018) dataset in 16 languages. Hungarian is not included among the 16 languages. It is intended to be used as a test set for cross-lingual transfer and the train set of MultiNLI is not translated.

5.4.3.3 Hungarian benchmarks

Unfortunately there are very few standardized non-English benchmarks although the list is growing with the Korean version of GLUE (Park et al., 2021) or Liro, a collection of Romanian benchmarks (Dumitrescu et al., 2021). The largest such collection for Hungarian is the Hungarian Language Understanding or HuLU benchmark (Ligeti-Nagy et al., 2022) introduced in January 2022. Our Hungarian experiments in Section 7.1 predate HuLU.

Unlike the English GLUE benchmark, very few high-level datasets are available for Hungarian, therefore downstream evaluation is very limited. In this section we describe the datasets most suitable for evaluating PLMs and we do not aim to give a complete overview of Hungarian NLP resources. György Orosz curated a list of Hungarian NLP tools and datasets[7].

The most notable public datasets are:

OpinHuBank (Miháltz, 2013) A sentiment analysis dataset with 10,000 sentences rated on a 5-scale.

Szeged Treebank The largest fully manually annotated treebank in Hungarian with 82,000 sentences.

Szeged Dependency Treebank (Vincze et al., 2010) A dependency-tree format version of the Szeged Treebank. It uses its own dependency labels which predate the Universal Dependencies standard.

Hungarian UD A subset of the Szeged Dependency Treebank converted to UD 1.0. Contains 1,800 sentences.

NerKor (Simon and Vadász, 2021) The largest gold standard named entity corpus for Hungarian with 1.5M sentences. Our main experiments on Hungarian (Ács et al., 2021) predate this dataset.

[7]https://github.com/oroszgy/awesome-hungarian-nlp

A	piacokért	folyó	világméretű	versenyt	a	fizetőképes
DET	NOUN	ADJ	ADJ	NOUN	DET	ADJ
kereslet	megjelenése	tette	lehetővé	.		
NOUN	NOUN	VERB	ADJ	PUNCT		

Table 5.2: Hungarian sentence with POS tagging.

WikiAnn (Pan et al., 2017) An automatically annotated named entity dataset in 282 languages including Hungarian.

Automatically tagged Webcorpus 2.0 (Nemeskey, 2020b) Webcorpus 2.0 is a large web crawl which was automatically analyzed with e-magyar (Sass et al., 2017). We use the POS tagging as it conforms with the UD UPOS tagging standard and it is much larger than the Hungarian UD treebank.

HuLU Hungarian Language Understanding. A new Hungarian benchmark based on the English GLUE benchmark. Our experiments on Hungarian (Ács et al., 2021) predate this dataset.

5.4.4 Part of speech tagging

POS tagging assigns a part-of-speech label to each token in a sentence. It is often required as an intermediary task between tokenization and dependency parsing. Silver and gold standard POS tagged datasets are available in most medium and high resource languages and many low resource languages as well. There are many different standards often crafted for specific languages. *Universal POS* or *UPOS*, a POS tagset introduced along with Universal Dependencies, is a cross-lingual standard which has been adapted to over 100 languages and is used in all UD treebanks. An example sentence is shown in Table 5.2.

5.4.5 Named entity recognition

Our other downstream evaluation task is named entity recognition (NER). NER identifies and classifies named entities. Depending the tagging standard, there are usually 4 types of NEs, person (PER), location (LOC), organization (ORG) and miscellaneous (MISC). Many named entities are multiword named entities, in other words, they span over multiple tokens. The inside-outside-beginning of IOB tagging schema (Ramshaw and Marcus, 1995) and its variant the IOB2 schema represent these as chunks in a sequence of tokens. The first token of a chunk is tagged `B-`, all continuation tokens are tagged `I-` and tokens not part of a named entity are tagged `O`. `B-` and `I-` are followed by the type of the entity such as `B-LOC`. Table 5.3 shows an example sentence with multiple multiword named entities.

Born	in	Winchester	,	Massachusetts	,	he	is	the
O	O	B-LOC	I-LOC	I-LOC	O	O	O	O
grandson	of	Major	League	Baseball	player	Lennie	Merullo	.
O	O	B-ORG	I-ORG	I-ORG	O	B-PER	I-PER	O

Table 5.3: An example sentence with named entities tagged using the IOB2 schema.

5.5 Large language models

BERT and other Transformer based models paved the way for scaling up in both model size and the amount of training data resulting in increasingly large models. OpenAI's GPT-2 (Radford et al., 2019), released in 2019, was the first major model with over a billion parameters. GPT-3 (Brown et al., 2020), its successor in 2020, was substantially larger with 175 billion parameters. It provided the base model for the hugely successful ChatGPT, an online chatbot with remarkable capabilities. As of November 2024 the latest edition is GPT-4o (OpenAI, 2023). The GPT-3 and GPT-4 are only available via APIs and the model architecture and weights were never released.

Llama (Touvron et al., 2023) is Meta's (formerly Facebook) autoregressive model family starting in February 2023. The latest version at the time of the thesis is Llama 3.2 released in September 2024. Unlike the GPT model family, Llama model weights are available for download.

LLMs standard usage is via *prompting* a.k.a. giving natural language instructions. Most LLMs are also chatbots with the ability to handle long multiturn conversations. They are also adept at multi-step reasoning (chain of thought). LLMs however are trained on general information and they lack the context for most industrial applications. *Retrieval augmented generation* or *RAG* is a general method for combining information retrieval with LLM prompting. The user provides their own specific documents, usually with the help of a vector database, to the LLM.

Although the main focus of this part of the thesis is the evaluation of the BERT model family, we extend our studies to LLMs albeit on a small scale. We acknowledge that the LLM results presented are limited in scope.

Chapter 6

Probing

Probing, also called *auxiliary prediction tasks* or *diagnostic classifiers*, is a popular method for analyzing PLMs. Probing aims to recover linguistic information from the representation extracted from a language model. A probing task is a small set of linguistically annotated data, typically a classification problem. This data is passed through the language model and the resulting representation is *probed* for the labels. A small adaptive layer, often a multilayer perceptron, is trained on top of this representation. If the classifier is able to predict the linguistic label, the model contains the data. The adaptive layer typically has orders of magnitudes fewer parameters than PLMs. The underlying model is generally not fine-tuned on the probing task but it may be fine-tuned on a downstream task (Ravichander et al., 2021).

6.1 Related work

Word embeddings dominated the mid 2010s NLP research and industry and continue to be essential building blocks of simple models. They were also studied directly via classifiers that examined various linguistic and other properties. The *embedding probes* (Luo et al., 2021) used simple classifiers on embedding vectors to recover certain semantic information such as color (Sommerauer and Fokkens, 2018), referential attributes (Gupta et al., 2015), among others. Rubinstein et al. (2015) compared different types classifiers on various taxonomical and attributive classification tasks. Köhn (2015) was perhaps the first work that extended this line of study to 6 other languages besides English. With the introduction of seq2seq models that encoded sentences rather than words, these studies provided a natural foundation to probing sentence embeddings.

The first probing on sentence encoders was performed by Köhn (2015) and Gupta et al. (2015), who trained classifiers on static word embeddings. The classifiers predicted various morphological, semantic and syntactic properties. To the best of our knowledge, Shi et al. (2016) was the first work that used probing classifiers on the encoder of trained neural machine translation models. They probed for two sentence level attributes, tense and voice as well as three phrase or word level attributes: POS, smallest phrase constituent and top-level syntactic sequence. Later work tended to simplify the probing target to sentence level attributes. Conneau et al. (2018a) probed for various sentence level attributes in trained seq2seq models. The models are trained on 4 kinds of seq2seq tasks: NMT (4 languages), AutoEncoder, SkipThought (Kiros et al., 2015) and grammatical parsing. Their probing tasks range from artificial tasks such as swapped bigram detection to attributes of the constituency parse tree and other syntactic properties. Zhang and Bowman (2018) used probing to compare 4 training objectives.

Belinkov et al. (2017a) was perhaps the first to probe for morphology in 6 languages including some morphologically rich languages in LSTM-based NMT models. They showed that character-based representations, particularly at lower layers, are better for such languages and higher morphological tagging accuracy correlates with higher BLEU scores. They also show that while the representations generated on the encoder side indeed contain morphology, the decoder lacks this ability. They hypothesize that the attention mechanism during decoding does the heavy lifting instead. They follow up with similar works for POS and syntactic probing (Belinkov et al., 2017b; Adi et al., 2017).

The next generation of probing papers targeted BERT and its variants among other models but they are mostly limited to English models. Hewitt and Manning (2019) showed that entire syntax trees can be recovered to some degree from BERT and ELMo models, while baseline models lack this quality. Liu et al. (2019a) probed BERT and ELMo for syntactic, POS and constituency labels. Tenney et al. (2019b) extend the set of probing task with some labeling tasks (NER, semantic role labeling, dependency labeling) and introduce a span directed probing setup named *edge probing*. They also include some semantic tasks but the probed models, BERT, ELMo, CoVe and GPT, fail to outperform the baselines on such tasks. Warstadt et al. (2019) focus on a single English-specific grammatical phenomenon, negative polarity items and find that BERT has some grammatical knowledge. Elazar et al. (2020) propose *amnesic probing*, a causal attribution method, which examines if the removal of certain information has a negative effect on a model's performance on some task.

Another line of investigation aims to find the location of certain linguistic information *within* BERT. Tenney et al. (2019a) claim that the English BERT *rediscovers* the classis NLP pipeline and lower layers are responsible for lower level tasks such as POS, while higher layers handle high level tasks such as coreference resolution. de Vries et al. (2020) argue that this layerwise separation is not as clear for English and Dutch models.

Despite its popularity, probing has been criticized for its proclivity to overestimate the model's capabilities. Voita and Titov (2020) showed that probing classifiers can 'learn' random labels to some degree over chance. Belinkov (2021) surveys all probing methods and points to several types of shortcomings. We address these shortcomings in Section 6.6.

6.2 Morphosyntactic probing

In this thesis, we focus on morphosyntactic probing. Like Shi et al. (2016), we include tense but we extend this set with 3 frequent morphosyntactic tags: case, gender and number. Unlike Belinkov et al. (2017a) we do not aim to perform full morphological tagging with a single task but rather focus on individual frequent tags. This reduces the probing classifier size and it makes the results easier to compare across languages.

UD provides annotation for over a 100 languages and these 4 tags are available in dozens of languages. Many well-resources languages use affixes to express these tags as well as agreement between certain parts of the sentences. Some languages have more limited morphology and most of the information is carried in prefixes and word order. English is one such example. Another group of languages uses templatic morphology.

Since morphological annotation is available in most UD Treebanks, we use morphological classification as the target of probing tasks. Morphology is traditionally defined as the study of a word formation and concepts covering multiple words or the full sentence are relegated to syntax. However, it is difficult to define the boundary between morphology and syntax, particularly in a way that works for multiple languages. English, for example, has very little morphology at the word level, much of the role that morphology plays in other languages, is relegated to syntax. The sentences *The cat is in the* *box* and *The cat is on the box* are different only in their preposition. Uralic languages have extensive case markings including multiple locative cases that would mark such a difference at the word level (*A macska a dobozban van* vs. *A macska a dobozon van.*

Universal Dependencies introduced a standardized morphological tagging schema that disregards the exact definition of morphology and simply uses word level morphological tags when applicable. The choice of tags to include in a treebank is up to the authors and it may differ between treebanks in the same language.

Table 6.1 shows an English sentence tagged using the CoNLL-U format. The sixth column contains each token's morphological tagging as a list of *Feature=Value* pairs. Although English does not have case inflection, most English treebanks do have *Case* features on nouns. The value set varies between {*Nominative*, *Accusative*} and {*Nominative*, *Accusative*, *Genitive*}, although *Genitive* is very infrequent.

Id	**Form**	**Lemma**	**UPOS**	**XPOS**	**Morphological features**
1	The	the	DET	DT	Definite=Def|PronType=Art
2	third	third	ADJ	JJ	Degree=Pos|NumType=Ord
3	was	be	AUX	VBD	Mood=Ind|Number=Sing|Person=3|Tense=Past|VerbForm=Fin
4	being	be	AUX	VBG	VerbForm=Ger
5	run	run	VERB	VBN	Tense=Past|VerbForm=Part|Voice=Pass
6	by	by	ADP	IN	_
7	the	the	DET	DT	Definite=Def|PronType=Art
8	head	head	NOUN	NN	Number=Sing
9	of	of	ADP	IN	_
10	an	a	DET	DT	Definite=Ind|PronType=Art
11	investment	investment	NOUN	NN	Number=Sing
12	firm	firm	NOUN	NN	Number=Sing
13	.	.	PUNCT	.	_

Table 6.1: Example sentence in CoNLL-U format. Columns related to dependency parsing are omitted for clarity.

6.2.1 Formulation

We define a probing task as a triple of ⟨language, POS, morphological feature⟩, following UD's naming conventions for morphological features. The task ⟨English, VERB, Tense⟩ requires identifying the tense of a verb (present or past) in its context. Each sample is a sentence with a particular target word and a morphological feature value corresponding to that word. The tense of *run* in the example in Table 6.1 can only be disambiguated in its context.

A probing dataset is a set of triples defined as

$$\mathcal{D} = \{s_i, t_i, l_i\}_{i=1}^{n}, \tag{6.1}$$

where sentence $s_i : w_1, \ldots, w_k$ is a sequence of $w_i, i \in [1, k]$ words, t_i is a target index and l_i is the morphosyntactic label of w_t, the tth token of sentence s_i. The dataset is always split into three parts, a train, a validation or development and a test set denoted as $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{dev}}, \mathcal{D}_{\text{test}}$ respectively. This means that although sentence context is available, the morphosyntactic tag itself refers to one token in the sentence. The same sentence may be part of multiple probing instances. The same is true for the target word forms, although we disallow the same form in more than one of the train, development and test sets.

The PLMs we probe use subword tokenizers that transform the input sentence into a list of subwords:

$$\text{Tokenizer}(s) = sw_1, \ldots, sw_{l,sw}, \tag{6.2}$$

where each sw_i subword is part of a predefined subword vocabulary. We also save the start position of each original token (start$_i$). In practice this is not supported natively by the tokenizers but we use the gold tokenization of UD and we run the subword tokenization on each token instead of the full sentence. This is very close to the strategy used by the tokenizers on full sentences. The starting positions are necessary for any token-level usage such as our morphosyntactic probing.

Since each token s_i may be segmented into multiple subwords, we need a way of pooling a single representation from multiple subword representations. We explore 9 options in Section 6.4 where we show that for morphology, the simplest strategies, namely taking the first or the last subword representation are just as good if not better than more sophisticated strategies. The index of the first subword of the target word is $j = \text{start}_t$ and the index of the last subword is $j = \text{start}_{t+1} - 1$. We always pick the better choice based on the development performance unless noted otherwise. Formally we pick subword function sp_k for probing task p as:

$$\arg\max_k \text{Acc}_p(\mathcal{D}_{\text{dev}}, \text{sp}_k), \tag{6.3}$$

where Acc is the accuracy of label prediction, i.e. the proportion of correctly classified samples and $\text{sp}_k \in \mathcal{SP}$ is the subword pooling function.

The other main component of our morphosyntactic probing is the *probing architecture* or *adaptive layer* itself. We use a small multilayer perceptron (MLP) in all experiments unless noted otherwise. The MLP has one hidden layer with 50 neurons. The input layer's size is the same as the hidden dimension of the model we probe, 768 in the case of BERT-base models. The output dimension is the number of classes in the probing task (see Section 6.3 for statistics on class d distribution). The label distribution of l_i is defined as:

$$P(l_i) = \sigma(\text{MLP}(\text{LM}(s_i)_j)), \tag{6.4}$$

where σ is the softmax function, LM is the language model we probe and j is the index of either the first or the last subword of the target word s_t. We use the most probable label $\arg\max(P(l_i))$ as the prediction of the probing classifier. The performance is evaluated using accuracy, the proportion of correctly classified examples.

6.2.2 Data sampling

Each token in UD is tagged by zero or more morphological features (e.g., case, number, gender, tense), including part of speech (POS). Every language, POS, and morphological feature comprise a candidate probe, with the task of predicting a given word token's feature value from the word and its context. Our sampling method takes sentences in the CoNLL-U format as its input and it samples each task separately with a few restrictions.

We avoid overlap of the target word in the train, validation and test splits, in other words one target word i.e. the word that is to be classified, can only appear in one these. UD treebanks have dedicated train, validation and test splits and we use these splits for our train, validation and test splits. We first merge all treebanks in a particular language and then sample the sentences in iteratively. Each sentence is sampled so that the target word does not appear as a target word in the other two splits.

Instead of sampling all train sentences right away, we sample 10 train sentences, then 1 validation and 1 test sentence. This avoids oversampling frequent target words in the train set.

Morphological value distributions are highly non-uniform. We limit class imbalance to 3:1, i.e., the largest class may not be more than 3 times larger than the smallest class. This requires downsampling frequent classes and in extreme cases, discarding very rare classes as happens with some Finnish noun cases.

Since the goal of probing is to find out what the model knows rather than what can be learned based on the probing data alone, small training sizes are desirable. We limit our training data to 2,000 train, 200 validation and 200 test samples (sentences). These numbers are achievable through a wide range of languages and tasks and they keep training times very short, thus allowing a very large number of experiments.

Our sampling method limits the sentence length between 3 and 40 tokens. The average sentence length is 20.5 tokens. The subword tokenizer of mBERT generates 38.2 tokens on average, while the subword tokenizer of XLM-RoBERTa outputs 34.2 tokens. Target fertility is defined as the number of subwords the target token is split into. Target fertility is 3.1 and 2.6 for mBERT and XLM-RoBERTa respectively. However, this measure varies substantially among languages, particularly in the case of mBERT's tokenizer. mBERT generates the fewest subwords for target words in English (2.05) and the most for Greek (4.75). XLM-RoBERTa on the other hand tends to split Persian the least (2.05) with English coming second (2.14) and it splits Hungarian the most (3.21) and the target fertility for all other languages is below 3 for XLM-RoBERTa. mBERT's target fertility is above 3 for 22 out of the 42 languages. This again suggests that XLM-RoBERTa's larger vocabulary is better suited for a multilingual setup.

Lastly we analyze the target words in more detail. Our data creation method disallows having the same target word appear in more than one of the train, validation or test splits. If the example sentence "I *read* your letter yesterday." is part of the train set and *read* is the target word, it may not appear as a target word in the validation or the test set regardless of its morphosyntactic analysis. It may be part of the rest of the sentence though. A probing task has 2024 unique target words on average. Split-wise this number is 1644 for the train split, 189 for the validation split and 190 for the test split. Recall that we have 2000 train, 200 validation and 200 test sentences. Interestingly there is very little ambiguity in the morphosyntactic analysis of the 4 tags we consider. 6.5% of words have ambiguous analysis in the full UD treebanks of the 42 languages.

6.2.3 Training details

Our model is illustrated in Figure 6.1. The input sentence is tokenized with a subword tokenizer. The subwords are passed to the language model which outputs a vector by layer for each subword. We then take the vectors corresponding to either the first or the last subword (subword choice is explored in Section 6.4) from each layer and use the weighted sum of these vectors (layer pooling options are explored in Section 6.6). This weighted sum is passed onto a small adaptive layer, a multilayer perceptron (MLP). This MLP and the layer weights are the only components we train, while the language model is frozen. Alternative approaches such as different MLPs and fine-tuning the model are explored in Section 6.6.

We train all classifiers with identical hyperparameters but independently of each other. There is no parameter sharing aside from the pre-trained model weights. The classifiers have one hidden layer with 50 neurons and ReLU activation. The input and the output layers are determined by the choice of language model and the number of target labels. This results in 40k to 60k trained parameters, far fewer than the number of parameters in any of the language models. All models are trained using the

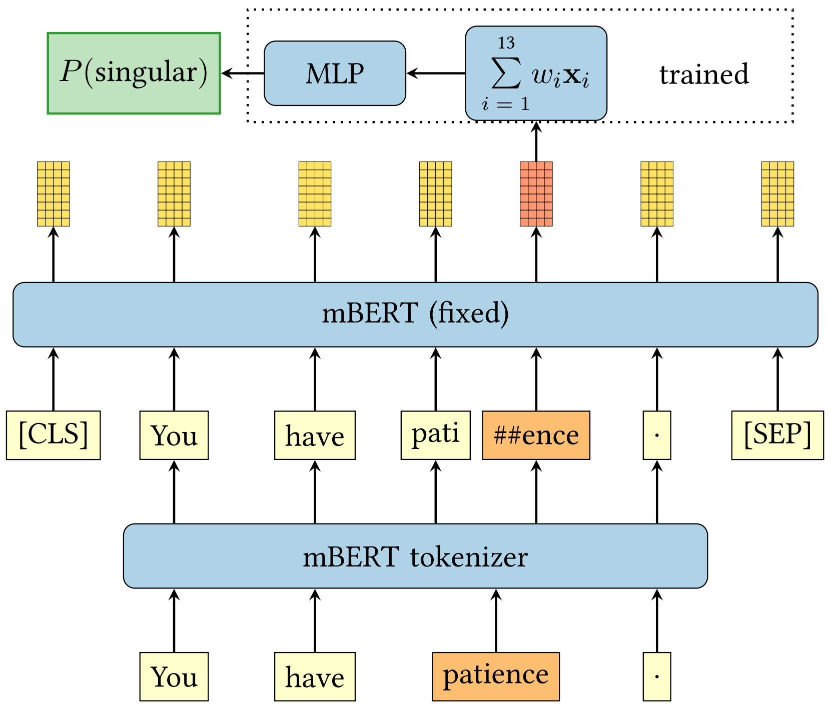

Figure 6.1: Probing architecture. Input is tokenized into wordpieces and a weighted average of the mBERT layers taken on the last wordpiece of the target word is used for classification by an MLP. Only the MLP parameters and the layer weights w_i are trained. x_i is the output of the ith layer, w_i is the learned layer weight. The example task here is ⟨English, Number, NOUN⟩.

Adam optimizer (Kingma and Ba, 2014) with $lr = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$. We use 0.2 dropout for regularization and early stopping based on the development set. The maximum number of epochs is set to 200 but in practice this is only reached in about 2% of the experiments. The batch size is always set to 128 except in the fine-tuning experiments where it is set to 8. All experiments are run on a single GPU, either a GeForce RTX 2080 Ti (12GB) or a Tesla V100 (16GB).

6.3 Multilingual probing dataset

We used the sampling method from Section 6.2.2 on the Universal Dependencies Treebanks to sample tasks in as many languages as possible.

6.3.1 Choice of languages and tags

UD 2.9 (Nivre et al., 2020) has treebanks in 122 languages. mBERT supports 104 languages while XLM-RoBERTa supports 100 languages. There are 55 languages in the intersection of these three sets. We include every language from this set except those where it is impossible to sample enough probing data. This was unfortunately the case for Chinese, Japanese, and Vietnamese due to the lack of data with morphosyntactic information and in Korean due to the different tagset used in the largest treebanks. 11 other languages have insufficient data for sampling. In contrast with e.g. Şahin et al. (2019), who used UniMorph for morphological tasks, a type level morphological dataset, UD allows studying morphology in context (often expressed through syntax). Moreover we extended UD 2.9

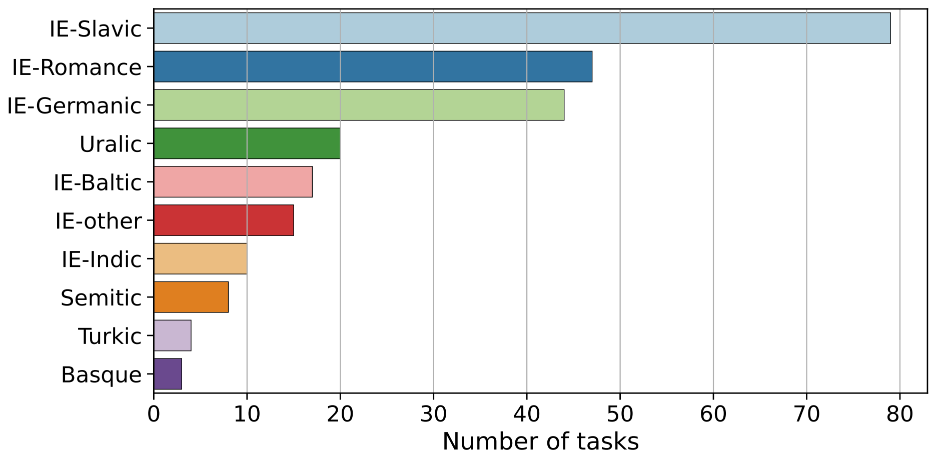

Figure 6.2: Number of morphosyntactic probing tasks by language family.

with Kote et al. (2019), an Albanian treebank and with a silver standard Hungarian dataset (Nemeskey, 2020a). The resulting probing dataset includes 42 languages.

UD has over 130 different morphosyntactic tags but most of them are only used for a couple of languages. In this work, we limit our analysis to four major tags that are available in most of the 42 languages: Case, Gender, Number, and Tense, and four open POS classes ADJ, NOUN, PROPN, VERB. Out of the $4 \times 4 = 16$ POS-tag combinations, 14 are attested in our set of languages. The missing two, ⟨NOUN, Tense⟩ and ⟨PROPN, Tense⟩, are linguistically implausible. One task, ⟨ADJ, Tense⟩ is only available in Estonian. The most common tasks are ⟨NOUN, Number⟩, ⟨NOUN, Gender⟩ and ⟨VERB, Number⟩, available in 37, 32 and 27 languages respectively. Figure 6.5 illustrates the availability of each task by language.

60% of the tasks are binary (e.g., ⟨English, NOUN, Number⟩), 20.6% are three-way (e.g., ⟨German, NOUN, Gender⟩) classification problems. The rest of the tasks have four or more classes. ⟨Hungarian, NOUN, Case⟩ has the most classes with 18 distinct noun cases, followed by ⟨Estonian, NOUN, Case⟩, ⟨Finnish, NOUN, Case⟩ and ⟨Finnish, VERB, Case⟩ with 15, 12, and 12 cases respectively. Figure 6.3 shows a histogram of the class count distribution.

Table 6.2 lists the 42 languages included in the probing dataset. The task counts vary greatly. We only have one task in Afrikaans, Armenian, and Persian, while we sample 13 tasks in Russian and 12 in Icelandic.[1] The resulting dataset of 247 tasks is highly skewed toward European languages as evidenced by Figure 6.2. Over 80% of the tasks come from the Indo-European family and the top 15 languages account to more than 50% of all tasks (Figure 6.4). The Slavic family in particular accounts for almost one third of the full dataset. This is due to two facts. First, Slavic languages have rich morphology so most POS/tag combinations exist in them (unlike, e.g., the Uralic languages which lack gender). Second, there are many Slavic languages and their treebanks are very large, the Czech treebanks are over 2M tokens, while the Russian treebanks have 1.8M tokens. The modest number of non-European tasks is an important limitation of our study. Fortunately the Indo-European language family is large and diverse enough that we have examples for many different morphosyntactic phenomena.

[1]The Icelandic UD was substantially expanded recently (Arnardóttir et al., 2020). We were not able to sample enough data from the earlier versions.

Family	Language	# Tasks	Family	Language	# Tasks
Basque	Basque	3	IE-Slavic	Belarusian	6
IE-Baltic	Latvian	11	IE-Slavic	Bulgarian	5
IE-Baltic	Lithuanian	6	IE-Slavic	Croatian	8
IE-Germanic	Afrikaans	1	IE-Slavic	Czech	10
IE-Germanic	Danish	3	IE-Slavic	Polish	10
IE-Germanic	Dutch	3	IE-Slavic	Russian	13
IE-Germanic	English	2	IE-Slavic	Serbian	7
IE-Germanic	German	10	IE-Slavic	Slovak	8
IE-Germanic	Icelandic	12	IE-Slavic	Slovenian	5
IE-Germanic	Norwegian Bokmal	4	IE-Slavic	Ukrainian	7
IE-Germanic	Norwegian Nynorsk	4	IE-other	Albanian	6
IE-Germanic	Swedish	5	IE-other	Armenian	1
IE-Indic	Hindi	6	IE-other	Greek	4
IE-Indic	Urdu	4	IE-other	Irish	3
IE-Romance	Catalan	6	IE-other	Persian	1
IE-Romance	French	7	Semitic	Arabic	4
IE-Romance	Italian	7	Semitic	Hebrew	4
IE-Romance	Latin	9	Turkic	Turkish	4
IE-Romance	Portuguese	6	Uralic	Estonian	7
IE-Romance	Romanian	6	Uralic	Finnish	8
IE-Romance	Spanish	6	Uralic	Hungarian	5

Table 6.2: List of languages and the number of tasks in each language.

6.3.2 Dataset statistics

Our sampling method limits the sentence length between 3 and 40 tokens. The average sentence length is 20.5 tokens. The subword tokenizer of mBERT generates 38.2 tokens on average, while the subword tokenizer of XLM-RoBERTa outputs 34.2 tokens. Target fertility (Ács, 2019) is defined as the number of subwords the target token is split into. Target fertility is 3.1 and 2.6 for mBERT and XLM-RoBERTa respectively. However, this measure varies substantially among languages, particularly in the case of mBERT's tokenizer. mBERT generates the fewest subwords for target words in English (2.05) and the most for Greek (4.75). XLM-RoBERTa on the other hand tends to split Persian the least (2.05) with English coming second (2.14) and it splits Hungarian the most (3.21) and the target fertility for all other languages is below 3 for XLM-RoBERTa. mBERT's target fertility is above 3 for 22 out of the 42 languages. Figure 6.6 shows the two models' fertility averaged over all samples in the same language. mBERT has a higher fertility for 34 languages out of 42 (left figure) but the picture is more extreme when we only look at the fertility of target words (right figure), where it is shown that mBERT has an often much higher target fertility for every language except English. This again suggests that XLM-RoBERTa's larger vocabulary is better suited for a multilingual setup.

We analyze the behavior of the tokenizers on one task, ⟨Hungarian, NOUN, Case⟩ in more detail. Hungarian has 18 noun cases and their inflection is highly regular. Each case, aside from the unmarked nominative, has a dedicated abstract morpheme that manifests as a suffix. There may be multiple versions of the surface form due to vowel harmony and assimilation but in general most cases can be described with two or three unique word endings. Table 6.3 lists the five most common subwords

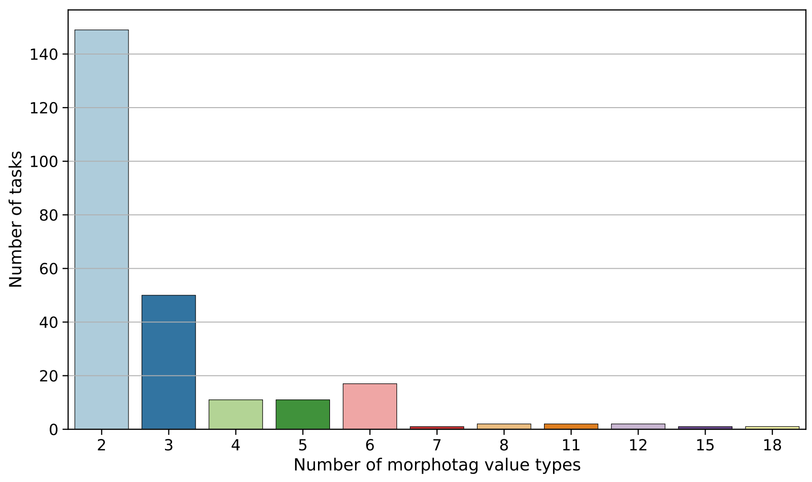

Figure 6.3: Number of binary, ternary etc. classes among the 247 morphology probing tasks.

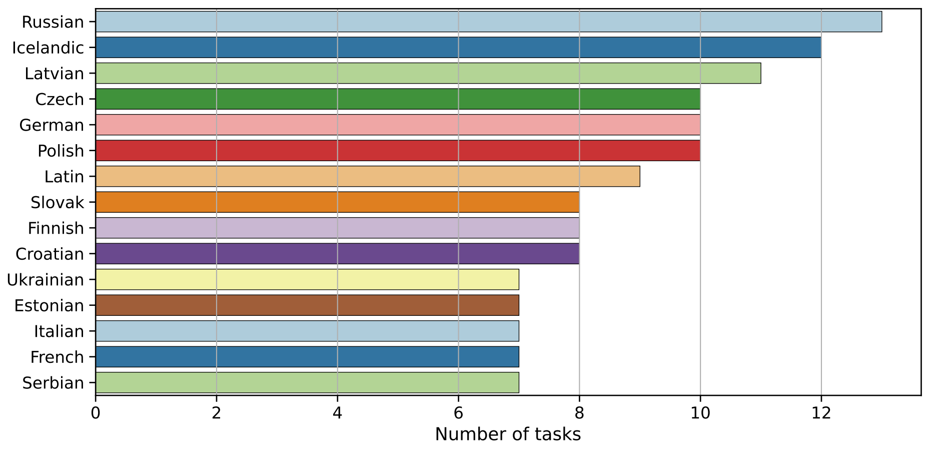

Figure 6.4: 15 languages with the most morphological probing tasks.

each tokenizer generates for each Hungarian case. Often the five most common subwords account for over 90% of the target words and in some cases such as the causative (Cau), there are fewer than five unique final subwords that the tokenizers ever use. Vowel harmony variations are also frequent in the table. While this analysis only concerns a single task, it is evident that the simple frequency-based subword tokenizers can pick up morphological regularities.

Next we analyze the target words in more detail. Our data creation method disallows having the same target word appear in more than one of the train, validation or test splits. If the example sentence "I *read* your letter yesterday." is part of the train set and *read* is the target word, it may not appear as a target word in the validation or the test set regardless of its morphosyntactic analysis. It may be part of the rest of the sentence though. A probing task has 2024 unique target words on average. Split-wise this number is 1644 for the train split, 189 for the validation split and 190 for the test split. Recall that we have 2000 train, 200 validation and 200 test sentences. Interestingly there is very little ambiguity in the morphosyntactic analysis of the 4 tags we consider. 6.5% of words have ambiguous analysis in the full UD treebanks of the 42 languages.

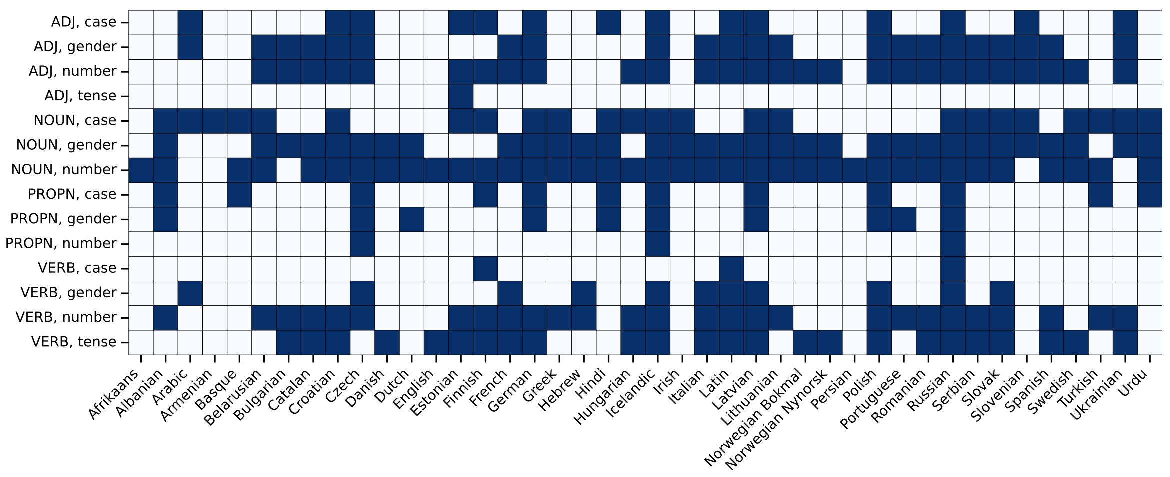

Figure 6.5: Task 'availability' by language. Blue indicates that a tasks is available in a language.

Label	mBERT	XLM
Abl	tól, ől, ól, től, ától (93.4%)	tól, től, étől, ától, októl (91.2%)
Acc	t, et, át, ket, ét (41.2%)	t, et, át, ét, okat (49.3%)
Ade	nál, nél, ál, él, l (99.3%)	nál, nél, ánál, eknél, énél (95.6%)
All	hoz, hez, oz, ához, éhez (86.0%)	hoz, hez, ához, khoz, éhez (71.3%)
Cau	ért, rt, t, RT (100.0%)	ért, áért, éért, T (100.0%)
Dat	nak, nek, ának, k, ének (77.9%)	nak, nek, ának, ének, ainak (75.0%)
Del	ról, ről, ől, ól, éről (85.3%)	ról, ről, éről, áról, ekről (78.7%)
Ela	ből, ból, ából, éből, ól (92.6%)	ből, ból, ából, éből, ekből (89.7%)
Ess	ként, ént, nt, ul, ül (91.9%)	ként, ul, ül, képp, sul (99.3%)
Ill	ba, be, ába, ébe, a (89.7%)	ba, be, ába, ébe, jába (91.2%)
Ine	ban, ben, ában, okban, jában (66.9%)	ban, ben, ában, okban, ében (74.3%)
Ins	ával, vel, val, l, kal (35.3%)	ával, sal, ekkel, ével, okkal (34.6%)
Nom	k, ás, és, ja, úra (14.7%)	ja, ok, t, k, ek (13.2%)
Sub	ra, re, ére, ára, ásra (74.3%)	ra, re, ére, ára, ekre (64.7%)
Sup	on, n, án, en, ján (58.1%)	on, án, en, kon, ján (54.4%)
Tem	kor, or, r, door, nr (99.3%)	kor, akor, or, skor, door (99.3%)
Ter	g, ig, áig, éig, . (90.4%)	ig, áig, tig, g, . (97.1%)
Tra	á, vé, vá, é, ává (72.1%)	ává, évé, sá, ké, vé (47.8%)

Table 6.3: The five most common last subwords by case in Hungarian noun cases. The overall frequency by case of the five most common last subwords is listed in parentheses.

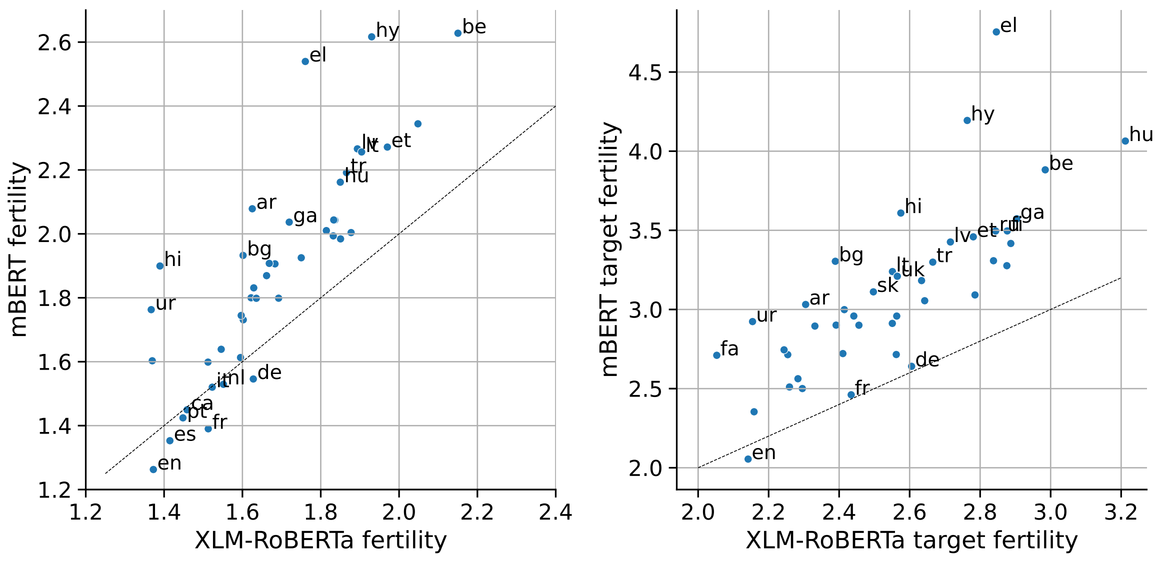

Figure 6.6: mBERT vs. XLM-RoBERTa fertility by language. The left plot shows the average fertility of the full probing dataset, the right plot depicts the average fertility of the target words. We added a dotted line at $x = y$ for illustration. Language codes are added for languages at the end of the two scales, the rest are omitted for clarity.

Since morphology is most often expressed as affixes in our languages, the target words tend to be longer than the average word length in characters. This also makes the tokenizers more likely to split the target words into multiple subwords. Target fertility is higher than sentence fertility in over 85% of the samples for both models. In 50% of the samples, the two models generate the same number of subwords, most often 2, for the target words. However, the remaining half is almost always (92%) split into fewer subwords by XLM-RoBERTa than mBERT. Figure 6.7 shows a heatmap of target subword counts for all samples and XLM-RoBERTa's tendency to generate one less subword is evident.

Lastly we look at the proportion of unsplit target words. It is reasonable to assume that an unsplit word has a better representation in PLMs than one that is split into multiple subwords that may or may not correspond to morphemes. Once again XLM-RoBERTa's larger vocabulary is less likely to split words and mBERT is more biased towards Indo-European languages, where the ratio of unsplit target words may be as high as 35% (⟨English, VERB, Tense⟩). Figure 6.8 shows a histogram of the proportion of unsplit tokens in a task by model. mBERT splits more than 90% of the target words in more than one third of the tasks, while XLM-RoBERTa is much less likely to do the same. Multiple subwords mean multiple BERT outputs that require some kind of pooling that creates a single tensor for a single token. In the next section, we explore how the choice of this pooling function affects model performance.

6.4 Subword pooling

BERT and other PLMs use subword tokenizers that generate one or more subwords for each token. This means that token-level usage of such models requires a way to pool multiple subword representations. Since most tasks consume full words, practitioners have the freedom to decide whether to use

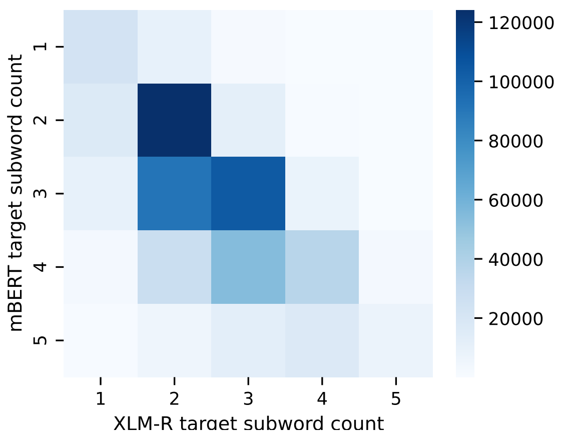

Figure 6.7: Heatmap of subword counts of the target word by model. Counts over 5 subwords (3.8% of all samples) are omitted for clarity.

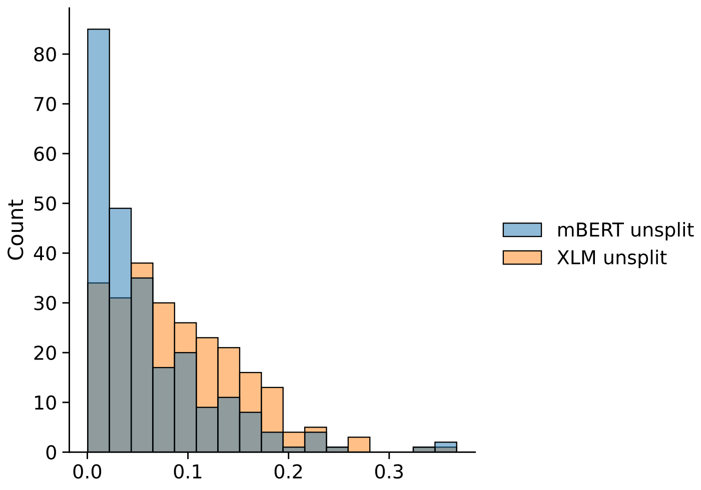

Figure 6.8: Histogram of the ratio of unsplit tokens in a task. Lower x-axis values mean that the proportion of multi-subword target words is higher.

the first, the last, or some combination of all subwords. The original paper introducing BERT, Devlin et al. (2019), suggests using the first subword for named entity recognition (NER), and did not explore different poolings. Kondratyuk and Straka (2019) also use the first subword, for dependency parsing, and remark in a footnote that they tried the first, last, average, and max pooling but the choice made no difference. Kitaev et al. (2019) report similar findings for constituency parsing, but nevertheless opt for reporting results only using the last subword. Hewitt and Manning (2019) take the average of the subword vectors for syntactic and word sense disambiguation tasks. Wu et al. (2020) use attentive pooling with a trainable norm for news topic classification and sentiment analysis in English. Shen et al. (2018) use hierarchical pooling for sequence classification tasks in English and Chinese.

In this section, we compare 9 subword pooling functions on 3 tasks in 9 languages. We run the experiments on two multilingual models, mBERT and XLM-RoBERTa, introduced in Section 5.2.2 and

Section 5.2.3 respectively. Both models have been extensively applied to English and multilingual tasks, but generally at the sentence or sentence pair level, where subword issues do not come to the fore. mBERT uses a common wordpiece vocabulary with 118k subword units. When a word is split into multiple subword units, each token that is not the first one is prefixed with ##. XLM-RoBERTa's vocabulary was trained in a similar fashion but with 250k units and a special start symbol (Unicode lower eights block) instead of continuation symbols. Each word is prefixed with this start symbol before it is tokenized into one or more subword units. These start symbols are often then tokenized as single units, particularly before Chinese, Japanese and Korean characters, therefore artificially increasing the subword unit count. We indicate the proportion of words starting with a standalone start symbol along with other tokenization statistics in Table 6.5.

As Table 6.5 shows, the number of subword tokens is highly dependent on the language. English words are only split in 14.3% (resp. 16.9%) of the time by the two models, while in many other languages more than half of the words are tokenized into two or more subword units. We hypothesize that this is due to the combination of the characteristics of the English language and its overrepresentation in the training data and the subword vocabulary.

We also observe that the two models' tokenizers work in very different ways. Out of the 2800 morphological test examples, only 58 are tokenized the same way and 51 of these are not split into multiple subwords. Only 7 words that are in fact tokenized, are tokenized the same way. Although the full tokenization is rarely the same, the first and the last subwords are the same in 45.5% and in 44.7% of the cases.

Our main contributions are:

- we show that subword pooling matters, the differences between choices are often significant and not always predictable;
- XLM-RoBERTa is slightly better than mBERT in the majority of morphological and POS tagging tasks; while mBERT is better at NER in all languages;
- the common choice of using the first subword is generally worse than using the last one for morphology and POS but the best for NER;
- the difference between using the first and the last subword is larger in lower layers than in higher layers and it is more pronounced in languages with rich morphology than in English;
- the choice of subword pooling makes a large difference for morphological and POS tagging but it is less important for NER.

This work was presented at the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL2021). The source code, the data and the full result tables are available on GitHub.[2]

6.4.1 Pooling methods

We test 9 types of pooling methods listed in Table 6.6 and grouped in three broad types. The first group uses the first and last subword representations in some combination. In F+L pooling the mixing weight is the only learned parameter. The second group are parameter-free elementwise pooling operations.

[2] https://github.com/juditacs/subword-choice

Language	**Tag**	**POS**	**# class**
Arabic	case	NOUN	3
Arabic	gender	ADJ	2
Czech	gender	ADJ	3
Czech	gender	NOUN	3
English	verbform	VERB	4
Finnish	case	NOUN	12
Finnish	verbform	VERB	3
French	gender	ADJ	2
French	gender	NOUN	2
French	verbform	VERB	3
German	case	NOUN	4
German	gender	ADJ	3
German	gender	NOUN	3
German	verbform	VERB	3

Table 6.4: List of morphological probing tasks we test the subword pooling methods on. The last column is the number of classes in a particular task.

	mBERT		**XLM-RoBERTa**		
	count	2+	count	2+	_start
Arabic	1.95	48.9	1.49	35.0	3.4
Chinese	1.58	53.5	2.13	88.5	86.6
Czech	2.04	53.0	1.7	45.2	1.6
English	1.25	14.3	1.25	16.9	0.8
Finnish	2.32	67.3	1.86	53.0	2.3
French	1.34	22.4	1.41	28.7	2.1
German	1.64	30.6	1.57	29.7	1.3
Japanese	1.6	43.0	2.25	94.6	92.9
Korean	2.44	75.7	2.16	67.3	9.0

Table 6.5: Subword tokenization statistics by language and model. First and third columns: average number of pieces that one word is split into. Second and fourth columns: proportion of multi-subword words. Last column: proportion of words that start with a standalone start token in XLM-RoBERTa.

Method	**Explanation**	**Params**
FIRST	first subword unit	none
LAST	last subword unit	none
LAST2	concatenation of the last two subword units	none
F+L	$wu_{\text{first}} + (1-w)u_{\text{last}}$	w
SUM	elementwise sum	none
MAX	elementwise max	none
AVG	elementwise average	none
ATTN	Attention over the subwords, weights generated by an MLP	MLP
LSTM	biLSTM reads all vectors, final hidden state	LSTM

Table 6.6: Subword unit pooling methods. u_{first} and u_{last} refer to the first and the last units respectively.

The last two methods rely on small neural networks that learn to combine the subword representations. Our subword ATTN has one hidden layer of 50 neurons with ReLU activation and a final softmax layer that generates a probability distribution over the subword units of the token. Similarly to self-attention, these probabilities are used to compute the weighted sum of subword representations to produce the final token vector. The LSTM uses a biLSTM (Hochreiter and Schmidhuber, 1997) that summarizes the 768-dimensional vectors (the hidden size of both models) into a 50-dimensional hidden vector in each direction, which are then concatenated and passed onto the classifier. These two are considerably more complicated and slower to train than the other methods, but ATTN works well for morphological tasks, and LSTM for POS tagging in CJK languages. Shen et al. (2018) found hierarchical pooling, a variety of our ATTN pooling method, beneficial, but they investigated sentence level tasks where the subword stream is much longer than in the word-level tasks we are considering (words are rarely split into more than 4 subwords) and hierarchical pooling has better traction.

6.4.2 Layer pooling effects

Both mBERT and XLM-RoBERTa have an embedding layer followed by 12 hidden layers. The only contextual information available in the embedding layer is the position of the token in the sentence via the positional encoding used in BERT models. Hidden activations are computed with the self-attention layers, therefore in theory have access to the full sentence. We ran our experiments for each layer separately as well as for the sum of all layers. For all tasks, as we move up the layers, results also move up or down in tandem. As exhaustive experiments considering different combinations of layers were computationally too expensive for our setup, and would significantly complicate presentation of our results, we pick a single setting for all experiments by computing the best *expected layer* for each task as

$$\mathbb{E}(L) = \frac{\sum_{l_i \in L} iA(l_i)}{\sum_{l_i \in L} A(l_i)}, \tag{6.5}$$

where L is the set of all layers, l_i is the ith layer, and $A(l_i)$ is the development accuracy at layer i. As Figure 6.9 shows, the expected layers are almost always centered around the 6th layer. Therefore, with the exception of comparing FIRST and LAST, which we analyze in greater detail in Section 6.4.4.1, we chose the 6th layer to simplify the presentation. This differs from our probing setup in that for morphosyntactic probing, we use the weighted sum of all layers as opposed to using a single layer. The reason is that for sentence-level tasks such as POS and NER, we cannot cache the output for the full sentence and all 13 layers due to memory constraints. Caching is essential for such a large number of experiments.

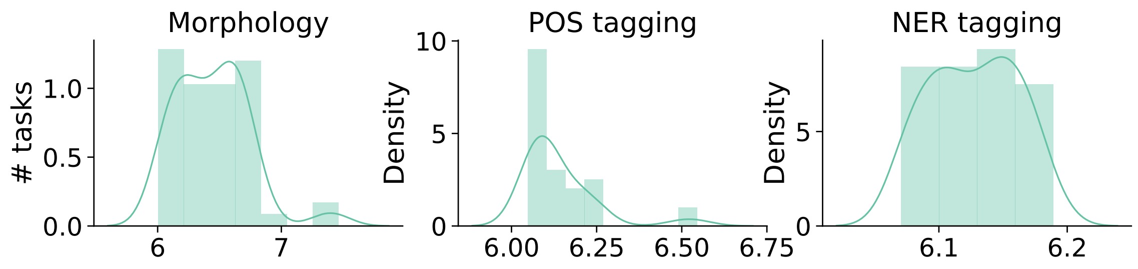

Figure 6.9: Distribution of the weighted average of layers across all tasks.

6.4.3 Experimental setup

We generally follow the training setup introduced in Section 6.2.3 for morphosyntactic probing. We train the same MLP architecture for POS tagging and NER on top of each token representation. We keep the number of parameters intentionally low, about 40k, to avoid overfitting on the probing data and to force the MLP to probe the representation instead of memorizing the data. We do note, however, that MLP and LSTM increase the number of trained parameters to 77k and 330k respectively. We run each configuration 3 times with different random seeds. The standard deviation of results is always less than 0.06 for morphology and less than 0.005 for POS and NER.

Choosing the size of the LSTM LSTM is our subword pooling method with the most parameters. The number of parameters scales quadratically with the hidden dimension of the LSTM. We pick this dimension with binary parameter search on morphology tasks. Our early experiments showed that increasing the size over 1000 showed no significant improvement, and a binary search between 2 and 1024 led us to choose a biLSTM with 100 hidden units.

6.4.4 Results

Our analysis consisted of two steps. We first performed the FIRST and LAST tasks at each layer (see Figure 6.10). Based on the results of this, we picked a single layer, the 6th, to test all 9 subword pooling choices. The full list of results on the 6th layer are listed in Tables 6.7, 6.10 and 6.11.

6.4.4.1 Layer pooling

We find that although LAST is almost always better than FIRST, the gap is smaller in higher layers. We quantify this with the ratio of the accuracy of LAST and FIRST at the same layer. Figure 6.10 illustrates this ratio for a few selected morphological tasks and POS and NER for all 9 languages. We split the morphological tasks into two groups, Finnish tasks and other tasks since the gap is so large in Finnish

task	**model**	FIRST	LAST	LAST2	F+L	SUM	MAX	AVG	ATTN	LSTM
⟨Arabic, Case, NOUN⟩	mBERT	**76.5**	73.3	74	74.8	75	73.5	75	75.5	74.8
⟨Arabic, Case, NOUN⟩	XLM-Ro	77.1	74.5	74	75.6	**80.9**	80.1	79.3	77.8	74.5
⟨Arabic, Gender, ADJ⟩	mBERT	93.3	97.3	97	97.5	98.8	98.2	98.2	**99.3**	97.3
⟨Arabic, Gender, ADJ⟩	XLM-Ro	96.5	97.5	97.2	97.5	99	99	99	**99.5**	97.5
⟨Czech, Gender, ADJ⟩	mBERT	41.5	**77.1**	74	72.6	64.2	63	63	75.6	68.0
⟨Czech, Gender, ADJ⟩	XLM-Ro	41.6	**73.5**	71.8	70.8	64.2	65	63.2	73	66.5
⟨Czech, Gender, NOUN⟩	mBERT	61.7	77.4	77.9	74.6	76.3	75.1	74.8	**81.3**	79.3
⟨Czech, Gender, NOUN⟩	XLM-Ro	69.3	**84.1**	83.3	83.7	80.6	82.4	81.1	82.9	82.4
⟨English, Verbform, VERB⟩	mBERT	83.7	**91.2**	87.2	89.2	89	89.5	89.7	90.7	88.5
⟨English, Verbform, VERB⟩	XLM-Ro	82.7	91	89.7	90.8	**91.8**	91.7	89	91.3	89.7
⟨Finnish, Case, NOUN⟩	mBERT	33	90.7	90.7	87.7	87.4	86.2	88.1	**93.9**	92.0
⟨Finnish, Case, NOUN⟩	XLM-Ro	49.6	94.4	94.7	94.2	94.9	95.7	95.7	**96.0**	95.4
⟨Finnish, Verbform, VERB⟩	mBERT	74.6	92.5	91.4	92.5	89.9	90	90.9	**93.2**	91.9
⟨Finnish, Verbform, VERB⟩	XLM-Ro	88.6	**97.8**	97.5	**97.8**	96.7	97.3	97.2	**97.8**	96.5
⟨French, Gender, ADJ⟩	mBERT	87.7	**93.2**	92.8	**93.2**	92.7	92.7	92.3	92.2	92.5
⟨French, Gender, ADJ⟩	XLM-Ro	83.2	92	91.7	90.5	89.7	89.3	88.7	**92.5**	90.3
⟨French, Gender, NOUN⟩	mBERT	88.3	**96.7**	94.7	95.2	95.3	95.2	96	95.8	96.2
⟨French, Gender, NOUN⟩	XLM-Ro	93.3	93.5	94.8	**96.0**	95.8	95.7	95.2	95.8	95.8
⟨French, Verbform, VERB⟩	mBERT	95.5	**100.0**	99.7	99.5	99.7	99.3	99.5	99.8	99.5
⟨French, Verbform, VERB⟩	XLM-Ro	93.5	99.3	99.2	99.2	**99.5**	**99.5**	99.3	99.3	99.0
⟨German, Case, NOUN⟩	mBERT	63.2	**77.7**	72	75.3	74	74.5	75.3	77	74.0
⟨German, Case, NOUN⟩	XLM-Ro	64.7	71.8	70.8	**77.3**	74.2	75	74.2	71.3	73.7
⟨German, Gender, ADJ⟩	mBERT	58.9	77.9	**78.1**	73.6	67.7	70	71.6	76.1	73.0
⟨German, Gender, ADJ⟩	XLM-Ro	54.7	74.6	**75.0**	74.5	73.8	71.8	71.8	74	69.7
⟨German, Gender, NOUN⟩	mBERT	60.5	79.1	78.8	78.3	77.8	78.6	78.4	**79.8**	76.8
⟨German, Gender, NOUN⟩	XLM-Ro	61.2	84.4	85.7	82.8	84.1	85.6	85.1	**88.9**	86.9
⟨German, Verbform, VERB⟩	mBERT	89.2	94.5	94.5	**95.0**	94.5	**95.0**	**95.0**	**95.0**	94.2
⟨German, Verbform, VERB⟩	XLM-Ro	90.4	94.4	94.7	94.2	94.4	93.9	94.4	**95.0**	94.7

Table 6.7: Full list of morphosyntactic probing results by subword pooling function at the 6th layer.

tasks that it would dwarf the other tasks on the same plot. ⟨Finnish, Case, NOUN⟩ shows the largest gap in the lower layers, LAST is 8 times better than FIRST. We observe smaller gaps in other tasks. POS shows a fairly uniform picture with the exception of Korean, where FIRST is worse in all layers and both models. Lower layers in mBERT show a larger gap in Czech and the same is true for Chinese and Japanese in XLM-RoBERTa. NER shows little difference between FIRST and LAST except for the first few layers, particularly in Chinese and Korean. To interpret these results, keep in mind that CJK tokenization is handled somewhat arbitrarily by XLM-RoBERTa, particularly in the first subword (cf. Table 6.5).

6.4.4.2 Morphology

We present the results of 14 morphological probing tasks (see Table 6.4) and 9 subword pooling strategies (see Table 6.6) using the 6th layer of each model.

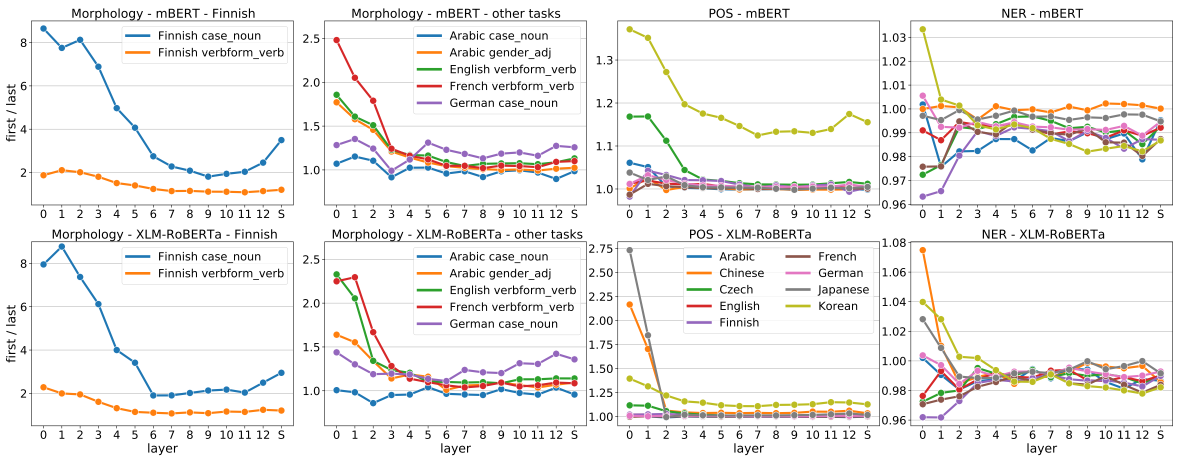

Figure 6.10: LAST-FIRST ratio of the test accuracy of some morphological tasks and of POS and NER in all languages across all layers. We plot Finnish morphological tasks separately since the effect is so pronounced that presenting them on the same plot would render the scaling uninformative for the other cases. S is the sum of all layers. Note that we do not have a strongly prefixing language due to the lack of available probing data.

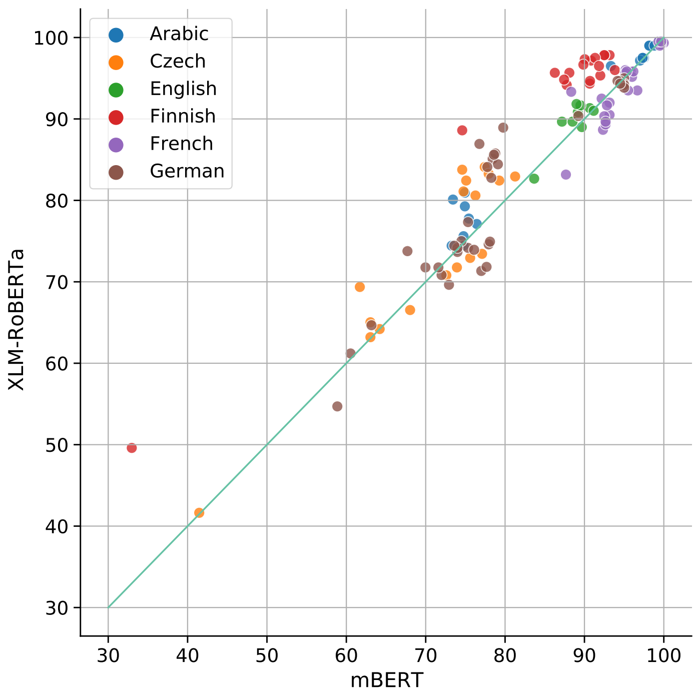

Figure 6.11: Accuracy of mBERT vs. XLM-RoBERTa on morphological tasks.

mBERT vs. XLM-RoBERTa Averaging over all tasks, XLM-RoBERTa achieves 85.7% macro accuracy while mBERT achieves 83.9%. On a per-language basis, XLM-RoBERTa is slightly better than mBERT in every language except for French. Figure 6.11 shows our findings. The two models generally perform similarly with the exception of French and Finnish: mBERT is almost always better at French tasks, while XLM-RoBERTa is always better at Finnish tasks. Similar trends emerge when looking at the results by subword pooling method. XLM-RoBERTa is always better regardless of the pooling choice but the difference is only significant ($p < 0.05$) for MAX and SUM.[3] These findings suggest that XLM-RoBERTa retains more about the orthographic presentation of a token, and it uses tokenization that is closer to morpheme segmentation, hence performing better at inflectional morphology, which is most often derivable from the word form alone.

First or last subword? As Figure 6.12 shows, with the exception of the ⟨Arabic, Case, N⟩ task, LAST is always better than FIRST. We find the largest difference in favor of LAST in Finnish and Czech. Table 6.8 lists all tasks where the difference between FIRST and LAST is larger than 20% along with the only counterexample (where the difference is about 10% in the other direction). These findings are likely due to the fact that Finnish and Czech exhibit the richest inflectional morphology in our sample.

The exceptional behavior of Arabic case may relate to the fact that case often disappears in modern Arabic (Biadsy et al., 2009). When this occurs the first token, being closest to the previous word, may provide a more reliable indicator, especially if that word was a preposition. Given the complex distribution of Arabic case endings, our sample is too small to ascertain this, and the results, about 75% on a 3-way classification task, are clearly too far from the optimum to draw any major conclusion (note that on Finnish case, a 12-way classification task, we get above 94%[4]).

Other pooling choices While FIRST is clearly inferior in morphology, the picture is less clear for the other 8 pooling strategies. As Figure 6.13 illustrates, ATTN is better than all other choices for both models but its advantage is only significant over a few other choices. We observe larger – and more often significant – differences in the case of mBERT than in XLM-RoBERTa.

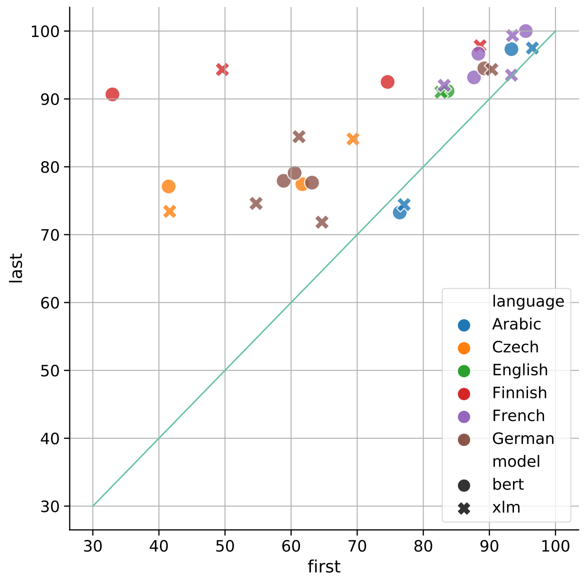

Figure 6.12: FIRST vs. LAST on morphological tasks.

[3]We use paired t-tests on the accuracy of the models on the 14 tasks.
[4]Finnish has more than 12 cases but infrequent ones were excluded.

Task	Model	FIRST	LAST
⟨Finnish, Case, N⟩	mBERT	33.0	90.7
⟨Finnish, Case, N⟩	XLM-RoBERTa	49.6	94.4
⟨Czech, Gender, A⟩	mBERT	41.5	77.1
⟨Czech, Gender, A⟩	XLM-RoBERTa	41.6	73.5
⟨German, Gender, N⟩	XLM-RoBERTa	61.2	84.4
⟨Arabic, Case, N⟩	XLM-RoBERTa	77.1	74.5
⟨Arabic, Case, N⟩	mBERT	76.5	73.3

Table 6.8: Morphological tasks with the largest difference between FIRST and LAST. The two tasks where FIRST is better than LAST are also listed.

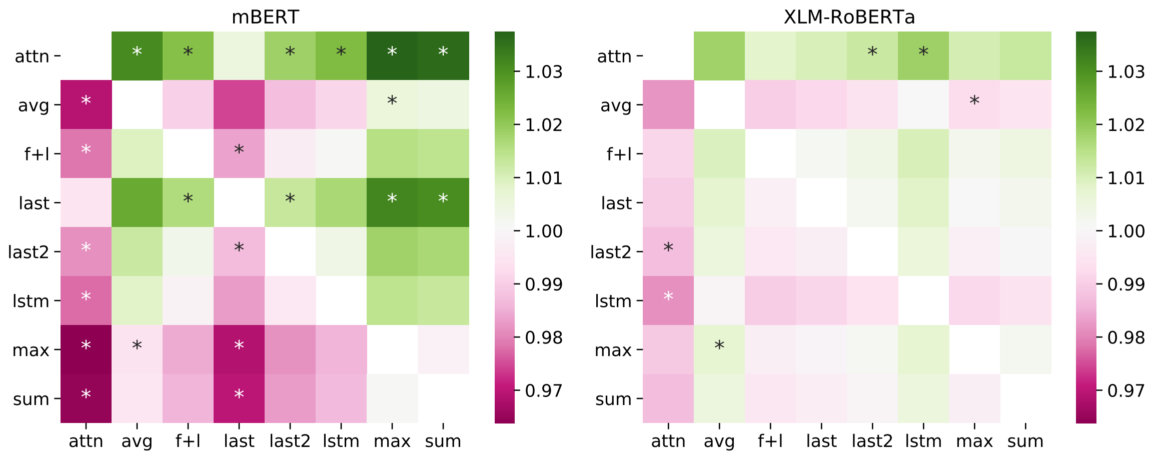

Figure 6.13: Pairwise ratio of test accuracy by subword choice on morphological tasks. Colors indicate row/column ratios. Green cells mean that the row subword choice yields better results than the column choice. $*$ marks pairs where the difference is statistically significant. ATTN is better than all other choices, therefore its row is green. FIRST is omitted for clarity as it is much worse than the other choices.

Attention weights The MLP used in ATTN assigns a weight to each subword which are then normalized by softmax. We examine these weights for each token in the test data for morphology. Table 6.9 lists the proportion of tokens where ATTN assigns the highest weight to the first, last or a middle token, or the token is not split by the tokenizer. The last subword is weighted highest in more than 80% of the cases. The only task where the last subword is not the most frequent winner is ⟨Arabic, Case, N⟩, where the first is weighted highest in 60% of the tokens by both models. These findings are in line with the behavior of FIRST and LAST.

6.4.4.3 POS tagging

We train POS tagging (cf. Section 5.4.4) models for 9 languages with 9 subword pooling strategies. We evaluate the models using tag accuracy. The full list of results is available in Table 6.10.

mBERT vs. XLM-RoBERTa As with morphological probing tasks, XLM-RoBERTa is slightly better than mBERT (95.4 vs. 94.6 macro average). We also observe that the choice of subword makes less difference than it does in morphological probing. Figure 6.14 shows that experiments in one language

	XLM-RoBERTa	mBERT
first	7.1%	6.0%
last	81.5%	83.7%
middle	5.9%	6.3%
single	5.5%	4.0%

Table 6.9: Distribution of the location of the highest weighted subword. Single refers to tokens that are not split by the tokenizer.

language	**model**	ATTN	AVG	F+L	FIRST	LAST	LAST2	LSTM	MAX	SUM
Arabic	mBERT	96.1	96.4	96.4	96.1	96.2	96.3	**96.5**	96.4	96.4
Arabic	XLM-RoBERTa	96.5	**96.8**	**96.8**	96.6	96.6	96.7	**96.8**	96.7	**96.8**
Chinese	mBERT	95.5	**96.2**	**96.2**	95.3	95.3	**96.2**	**96.2**	96.1	96.1
Chinese	XLM-RoBERTa	94.9	95.3	**95.4**	92.6	95.3	**95.4**	**95.4**	95.1	**95.4**
Czech	mBERT	98.2	98.2	**98.3**	97.5	98.2	**98.3**	**98.3**	98.2	98.1
Czech	XLM-RoBERTa	98.4	98.4	98.4	98.0	98.3	98.4	**98.5**	98.3	98.4
English	mBERT	95.8	**96.0**	**96.0**	95.5	95.7	95.8	**96.0**	95.8	95.7
English	XLM-RoBERTa	96.1	96.3	96.2	95.9	96.1	96.3	**96.4**	96.3	96.2
Finnish	mBERT	90.9	91.4	91.1	90.1	90.2	91.6	**92.3**	91.2	91.2
Finnish	XLM-RoBERTa	94.8	95	95	94.3	94.1	94.9	**95.3**	94.9	95
French	mBERT	97.4	**97.6**	97.5	97.4	97.4	97.4	97.4	97.5	**97.6**
French	XLM-RoBERTa	97.6	**97.7**	**97.7**	97.5	97.6	97.6	97.6	97.6	97.6
German	mBERT	97.8	97.9	**98.0**	97.4	97.9	**98.0**	**98.0**	97.9	97.9
German	XLM-RoBERTa	97.7	97.9	97.9	97.5	97.9	97.9	**98.0**	97.9	97.8
Japanese	mBERT	96.0	96.5	96.6	95.7	95.9	96.5	**96.8**	96.4	96.3
Japanese	XLM-RoBERTa	96.4	96.8	96.7	95.2	96.6	96.8	**97.0**	96.7	96.7
Korean	mBERT	94.1	93.7	94.3	84.8	93.6	94.3	**94.5**	93.7	93.8
Korean	XLM-RoBERTa	94.9	94.7	95	87.6	94.3	95	**95.1**	94.5	94.7

Table 6.10: Full list of POS tagging results by subword pooling function at the 6th layer.

tend to cluster together regardless of the subword pooling choice except for a few outliers: FIRST for Chinese and Korean is much worse in both models. The same result can be observed in Japanese, to a lesser extent though. Language-wise we find that XLM-RoBERTa is much better at Finnish and somewhat worse in Chinese but the two models generally perform similarly.

Choice of subword. As with morphology FIRST the is the worst choice, but the effect is not as marked for POS tasks. In Figure 6.14 we observe 3 outliers, XLM-RoBERTa, FIRST for Chinese and FIRST for Korean for both models. The only consistent trend is that XLM-RoBERTa is clearly better for Finnish regardless of the choice of subword pooling. The picture is less clear for other languages.

We split the analysis into CJK and non-CJK languages. Figure 6.15 and Figure 6.16 show a comparison for non-CJK languages and CJK languages respectively. The difference between choices is generally much smaller than for morphology. FIRST is the worst choice both for CJK and non-CJK languages. Interestingly one of the best choices for morphology, LAST, is the second worst choice for POS tagging, while one of the worst for morphology, LSTM, is one of the best for POS tagging. We hypothesize that this is due to overparametrization for morphology. POS tagging is a much more complex task that needs a larger number of trainable parameters (recall that LSTM parameters are shared across all tokens).

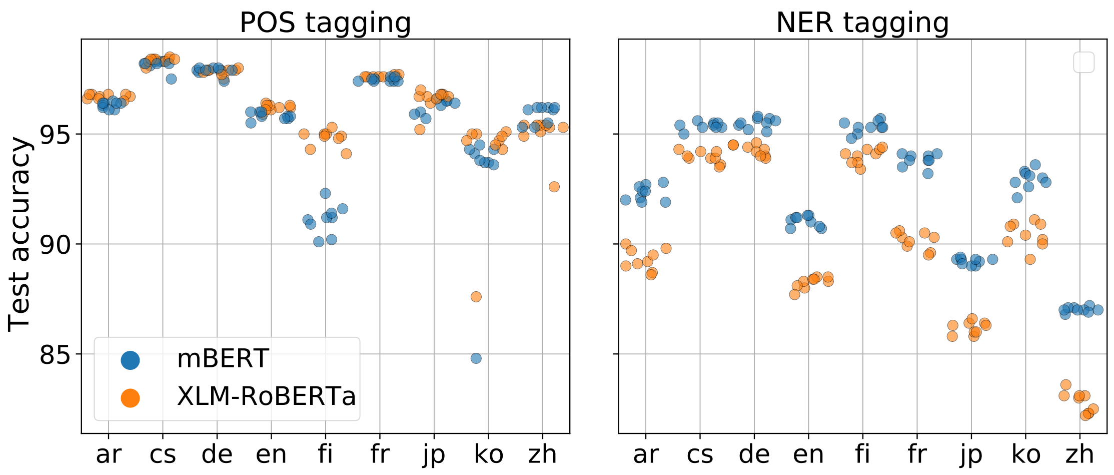

Figure 6.14: mBERT vs. XLM-RoBERTa for POS tagging and NER.

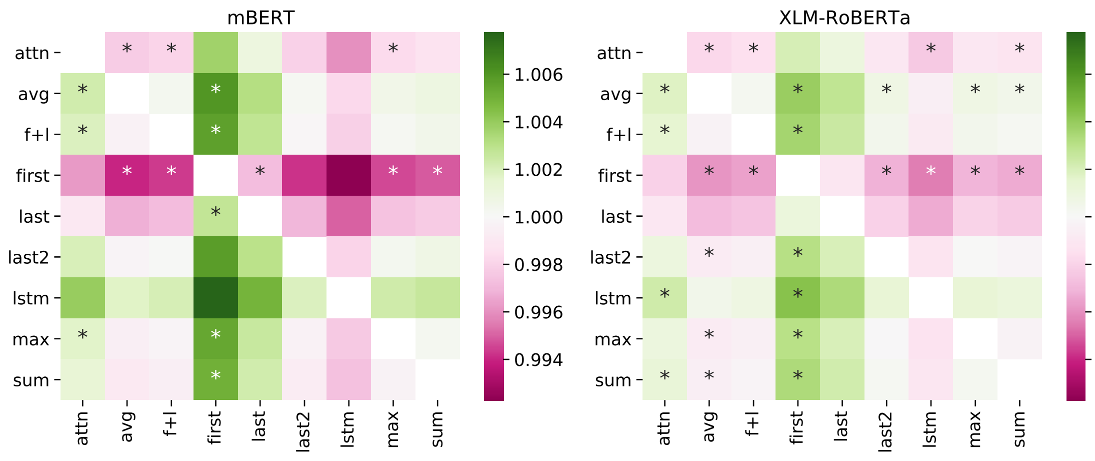

Figure 6.15: Subword choices for POS tagging in non-CJK languages. See Figure 6.13 for an explanation of the figure. FIRST is included.

6.4.4.4 Named entity recognition

As Figure 6.14 shows, in NER (cf. Section 5.4.5) the choice of subword pooling makes far less difference than in morphology. In terms of models, mBERT has a clear advantage over XLM-RoBERTa when it comes to NER. The difference between the two models is generally larger than the difference between two subword choices within the same language. The smallest difference between the two models appears to be in Czech, Finnish and German, which all have rich, partially agglutinative, morphology. This fits with our earlier findings that showed that XLM-RoBERTa might be better at handling rich morphology. Overall FIRST and the related F+L as well as LSTM come out as winners, the differences are rather small and often not statistically significant for CJK. The full results are listed in Table 6.11.

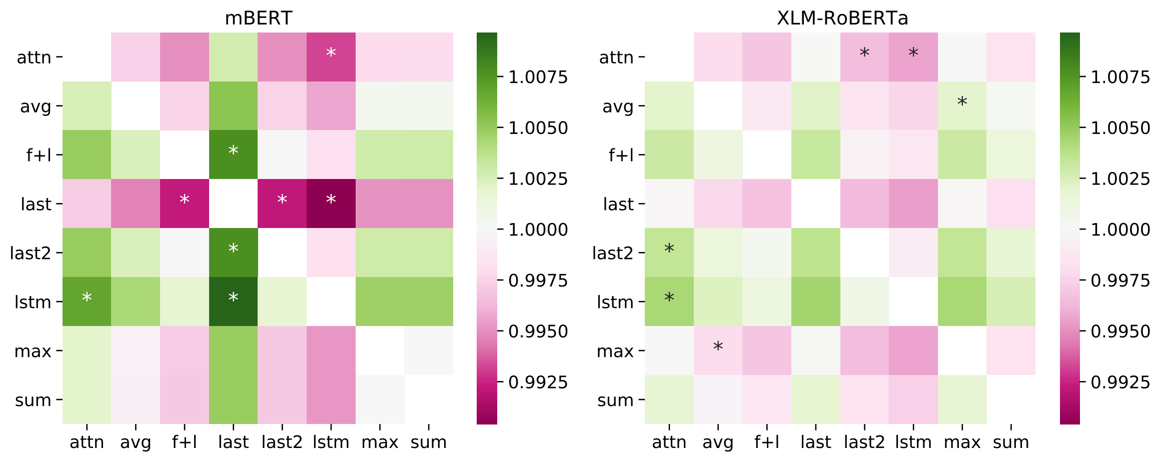

Figure 6.16: Subword choices for POS tagging in CJK languages. See Figure 6.13 for an explanation of the figure. FIRST is omitted for clarity as it is much worse than the other choices.

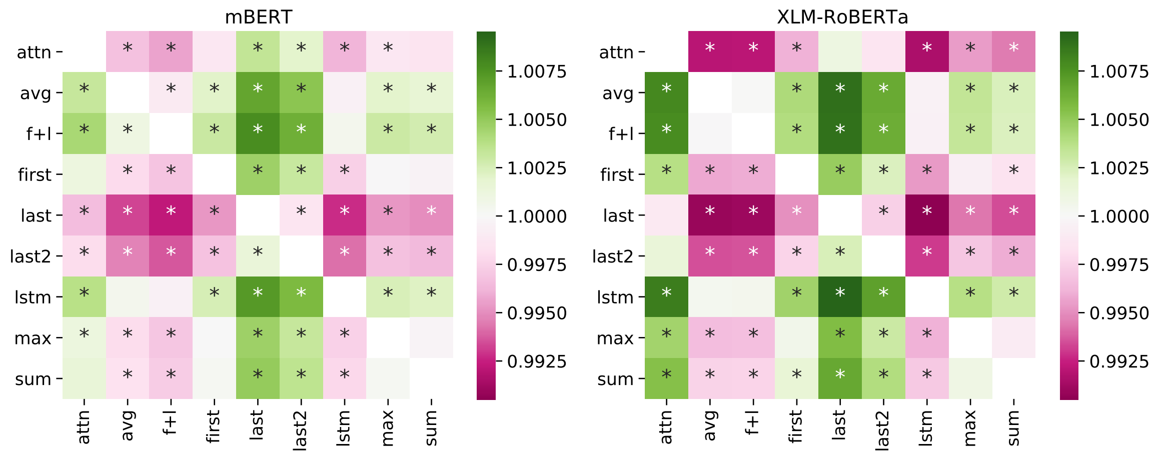

Figure 6.17: Subword choices for NER tagging in non-CJK languages. See Figure 6.13 for an explanation of the figure. FIRST is included.

6.4.5 Discussion

Throughout our extensive experiments we observed that pooling strategies can have a significant impact on the conclusions drawn from probing experiments. When considering multiple typologically different languages, the strength of the conclusions drawn from experiments can be weakened by considering a single pooling option. Our recommendation for NLP practitioners is to try at least three subword pooling strategies, particularly for tasks in languages other than English. FIRST and LAST usually gives a general picture – as a third control we recommend ATTN and LSTM. More complicated tasks such as POS or NER tagging may require LSTM with many parameters, while tasks that rely more on the orthographic representation such as morphology tend to benefit from ATTN.

Our methodology is only limited by the availability of data. It would be interesting to extend these study with languages that use prefixes too such as Indonesian or Swahili.

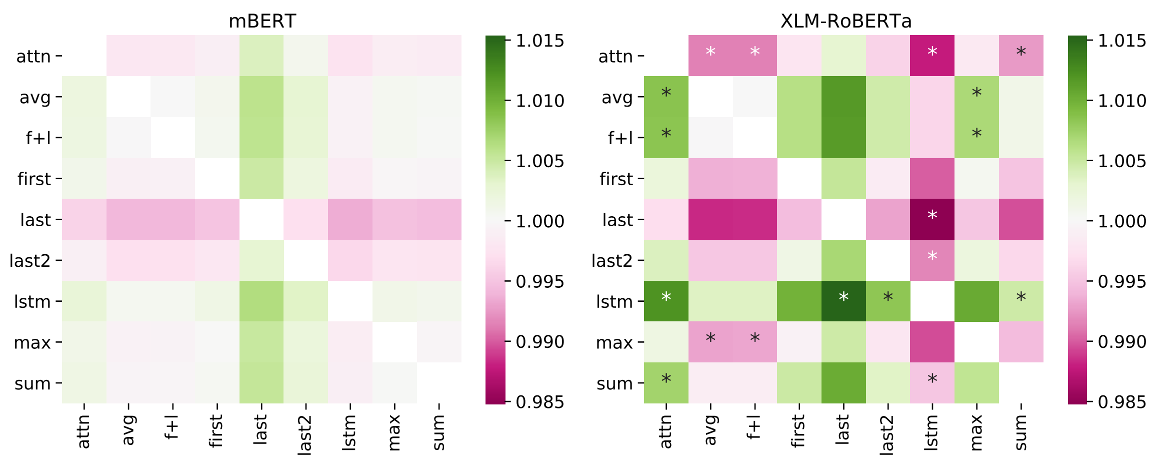

Figure 6.18: Subword choices for NER tagging in CJK languages. See Figure 6.13 for an explanation of the figure. FIRST is omitted for clarity as it is much worse than the other choices.

language	**model**	ATTN	AVG	F+L	FIRST	LAST	LAST2	LSTM	MAX	SUM
Arabic	mBERT	92.0	92.6	**92.8**	92.4	91.9	91.9	92.7	92.1	92.4
Arabic	XLM-RoBERTa	88.6	89.8	89.7	89.1	88.7	89.0	**90.0**	89.2	89.5
Chinese	mBERT	87.0	87	87	87	86.9	87.1	86.8	87.1	**87.2**
Chinese	XLM-RoBERTa	82.2	83.1	83.1	82.5	82.3	83.1	**83.6**	82.3	83
Czech	mBERT	95.3	95.5	**95.6**	95.3	95.0	95.3	95.5	95.4	95.4
Czech	XLM-RoBERTa	93.6	94.2	**94.3**	93.9	93.5	93.9	94.2	93.9	94
English	mBERT	90.8	91.2	**91.3**	91	90.7	90.7	**91.3**	91.1	91.2
English	XLM-RoBERTa	87.7	**88.5**	**88.5**	88.3	88.0	88.1	88.4	88.3	88.4
Finnish	mBERT	95.3	95.5	95.6	95.3	94.8	95.0	**95.7**	95.3	95.3
Finnish	XLM-RoBERTa	93.7	**94.4**	94.3	94	93.4	93.7	94.3	94.1	94.1
French	mBERT	93.8	**94.1**	**94.1**	93.8	93.2	93.5	94	94.0	93.8
French	XLM-RoBERTa	89.9	90.5	**90.6**	90.1	89.5	89.6	90.5	90.3	90.3
German	mBERT	95.4	95.7	**95.8**	95.6	95.1	95.2	95.7	95.5	95.5
German	XLM-RoBERTa	94.0	94.5	94.5	94.3	93.9	94.0	**94.6**	94.2	94.4
Japanese	mBERT	89.2	89.3	89.3	**89.4**	89.0	89.0	89.3	89.3	89.1
Japanese	XLM-RoBERTa	85.8	86.4	86.4	86.3	85.8	86.0	**86.6**	86.0	86.3
Korean	mBERT	92.8	93.3	93.2	92.8	92.1	92.6	**93.6**	93.0	93.1
Korean	XLM-RoBERTa	90.2	90.9	90.9	90	89.3	90.1	**91.1**	90.4	90.8

Table 6.11: Full list of NER results by subword pooling function at the 6th layer.

The key takeaway from this section is that performance on lower level tasks depends on the way we pool over multiple subword units that belong in a single word token. This is more of an issue in languages other than English, where a significantly larger proportion of words are represented by multiple subword units.

Morphological and POS tasks are both probing word-level attributes, but the results show huge disparity: for the morphological tasks FIRST pooling is the worst strategy, and ATTN is the best, while for POS tagging ATTN is almost as bad as FIRST, the best being LSTM. The NER task is intermediary between word- and phrase-level, and subword pooling effects are less marked, but still statistically significant.

6.5 Morphology in pre-trained language models

In this section examine how well morphology can be recovered from the model representations. We first compare the models with various baselines in Section 6.5.1 including pre-trained fastText vectors and multiple LSTM variations trained solely on the probing data. We also probe the large version of XLM-RoBERTa named *XLM-L* or *XLM-large* and mT5 (Xue et al., 2021), a massively multilingual generative model. In Section 6.5.3 we focus on the two large language models in comparison with the strongest baseline and we analyze the results along various categories. Finally we compare mBERT and XLM-RoBERTa in Section 6.5.4. As we showed in Section 6.4.4.2 the choice of subword pooling makes very little difference in morphosyntactic tasks, we opt to use the fastest options, we either pick the first or the last subword of the token in question. Exploring all options for all languages and tasks would be prohibitively expensive.

This work was published in the Journal of Natural Language Engineering (Ács et al., 2023). Some parts of this and the following sections are borrowed from Ács et al. (2023) but we also extended the experiments with newer models, most notably GPT-4o and its smaller version GPT-4o-mini.

6.5.1 Baselines

Our baseline models are described in Table 6.12 by three attributes. The first is pre-training. The probed language models, mBERT and XLM-RoBERTa are both pre-trained on vast datasets. Our only pre-trained baselines is fastText (Grave et al., 2018), a character n-gram based model trained on Wikipedia and Common Crawl similarly to XLM-RoBERTa[5]. The rest of the baselines are LSTM models trained solely on the train set of a particular probing task.

The second attribute is whether a model is contextual or not. Contextual models use the full probing sentence during both training and inference time. In contrast, non-contextual models only use the target word form and ignore its context. In theory the latter type is incapable of distinguishing between syncretic forms[6]. Models marked as *target only* rely solely on the target word. Although fastText was trained on full sentences, the vectors themselves are context independent.

Our last attribute is the choice of tokenization. mBERT, XLM-RoBERTa and other PLMs use subword tokenization with their own vocabulary (cf. Section 5.3). We borrow mBERT’s vocabulary and tokenizer and apply it to LSTM models as well. We compare these with character LSTM models. It should be noted that subword LSTM models require about ten times more parameters than character LSTM models since their input vocabulary and hence their embedding is much larger. We note the parameter count of the probing layer (not the LM itself) in the last column.

6.5.1.1 Training details

The MLP on top of mBERT and XLM-RoBERTa is intentionally small and we match the parameter number in chLSTM with a character embedding with 30 dimensions and a bidirectional LSTM with the hidden dimension set to 100 in total (the exact numbers are listed in Table 6.12). We performed a small scale hyperparameter search for the LSTM parameters and it turns out that the results are largely unchanged unless we use very small LSTMs and embeddings. Subword models need a considerably larger embedding in order to encode the much larger input vocabulary. We set this to 768 following the embedding size of both mBERT and XLM-RoBERTa. We also set the LSTM size to 768. These

[5]Both Wikipedia and Common Crawl are ongoing projects and XLM-RoBERTa and fastText were released a few years apart. This could result in substantial difference in data size.

[6]Different inflections that happen to have the same surface form.

result in millions of parameters as opposed to the modest number of parameters used in chLSTM. XLM-large has more trainable parameters because its hidden dimension is 1024 as opposed to 768 in mBERT, XLM-RoBERTa and mT5.

The embedding input size is determined by the input vocabulary size. Character models use a very small (usually < 100) input vocabulary, while subword models use inventories with over 100,000 elements. Our baseline models that use subwords only use the subset that appears in the train part of the probing data but this still results in thousands of distinct subwords. Target only models have fewer parameters because the input vocabulary is smaller than in models that use the full sentence.

Model	**Pre-trained on**	**Contextual**	**Tokens**	**Params**
mBERT	100 largest Wikipedia	yes	subword	38k
XLM-RoBERTa	Wikipedia + Common Crawl	yes	subword	38k
XLM-Large	Wikipedia + Common Crawl	yes	subword	51k
XLM-MLM-100	Wikipedia + Common Crawl	yes	subword	64k
DistilmBERT	distilled mBERT	yes	subword	38k
mT5	Common Crawl	yes	subword	38k
char LSTM	none	yes	character	42k
subword LSTM	none	yes	subword	8M
char LSTM - target only	none	no	character	40k
subword LSTM - target only	none	no	subword	4.8M
fastText	Wikipedia + Common Crawl	no	n-gram	15k
Stanza	UD	yes	mixed	N/A
GPT-4o	N/A	yes	subword	200+ billion
GPT-4o-mini	N/A	yes	subword	N/A

Table 6.12: Attributes of the probed models. The last column is the number of trainable probing parameters. It does not include the frozen parameters of the pre-trained model. Stanza models are not trained on the probing data.

6.5.1.2 Stanza

Stanza is a collection of linguistic analysis tools in 70 languages trained on UD treebanks. Stanza is an ideal tool for comparison since it was trained on the same dataset we use to sample our probing tasks. The underlying model is a highway bidirectional LSTM (Srivastava et al., 2015) with inputs coming from the concatenation of Word2vec or fastText vectors, an embedding trained on the frequent words of UD and a character-level embedding generated by a unidirectional LSTM. We use the default models available for each language and we do not fine-tune the models.

It works in a pipeline fashion and one of its intermediate steps is morphosyntactic analysis in UD format. We use this analysis as a high quality baseline for all of our 42 languages and 247 tasks except Albanian (6 tasks). We apply Stanza on the full probing sentences and we extract the morphosyntactic analysis of the target words. Since Stanza's own tokenizer often outputs a different tokenization than UD's gold tokenization, we extract every overlapping token and check the union of the morphosyntactic tags if there is more than one such overlapping token. This results in a set of `Feature=Value` pairs. We extract each occurrence of the feature of the particular probing task. If there is only one and it is the same as the reference label, it is correct. If there are more than one values and the correct value is among them, we divide one by the number of distinct values. In practice, this is very rare, 91% of the time, there is only one value in it is the same as the reference value. 8.3% of the time the reference value is not in the analysis of Stanza. The remaining 0.7% are the cases where there are multiple values including the *correct* one.

6.5.1.3 GPT-4o and GPT-4o mini

GPT-3 and later models show remarkable success in a variety of high-level tasks in many languages. Although these large LLMs are not the main focus of this thesis, we perform a small test on GPT-4o and its smaller variant GPT-4o-mini. The exact technical details of these models, and the difference between the two, are not publicly available. Both models are available via paid APIs.

Although there are various open source LLMs, we opt to use GPT-4o for one main practical reason: it supports *structured outputs*, i.e. we can restrict its output to be one of the desired morphosyntactic labels. Our preliminary tests with GPT-3.5 and Llama-8B showed low adherence to the 'rules' regardless of the phrasing of the prompt. GPT-4o accepts enumeration as a JSON schema.

We use GPT-4o without fine-tuning (zero-shot setup) and every prompt is run separately in batch mode. We run a single prompt for every test sample, 49,567 in total. Since the details of GPT's training data are not publicly available, it is possible that UD's test set is part of the training set.

Fortunately GPT-4o has a batching mode with support for JSON schema along with the templates. We tried a few variations for the exact prompt but we did not perform extensive prompt engineering. To level the playing field, the prompt includes every information available for the traditional probes. The prompt template is defined below and for each test sample, we replace the values denoted in curly braces.

Morphosyntactic prompting template

```
System message:
You are a linguistic expert.

User message:
### Instruction
Below is a sentence in {language}.
What is the {tag} of the {POS} {token}?
It is the {i}th token of the sentence.

### Sentence
{sentence}

### Answer:
```

6.5.2 Results

This section presents our morphosyntactic probing results across 247 tasks in 42 languages. We greatly extend the original experiments of Ács et al. (2023) with 6 additional pre-trained models and we prompt GPT-4o and GPT-4o-mini with the test set of our probing dataset. We also add new baselines.

First we note some general observations, then we analyze the behavior of mBERT and XLM-RoBERTa in comparison with the best baseline, chLSTM.

PLMs are on par with Stanza. Stanza is a high quality specialized library with language specific models. The best performing pre-trained language models are on par with the accuracy of Stanza. It should be noted that Stanza is a full featured NLP toolchain and we only compare one of its functionalities with PLMs. It is also much faster and better suited for large-scale analysis.

chLSTM is the best baseline. Table 6.14 lists the average accuracy of the 14 tasks by model as well as the average of the 247 tasks for each model. Figure 6.19 illustrates this grouped by POS. It is clear that both mBERT and XLM-RoBERTa are significantly better than any of the baselines including fastText but the margin is different. chLSTM, a character LSTM trained on the full sentence, is the best baseline with 85% accuracy on average and it provides a practical upper estimate of what is *learnable* from the probing data alone. fastText is a close second with a 83.3% average.

Contextual models are better than non-contextual models. Context indeed plays a role in morphosyntactic probing as evidenced by the 2.2% point difference between chLSTM and chLSTM-t, the latter only using the target. The subword versions of these models swLSTM and swLSTM-t are 4.5% point different. Verbal tasks rely less on the context while we see a bigger gap in nominal and adjective tasks. This is unsurprising given that verbal morphology tends to be exhibited on the word form rather than the context. The differences being relatively small suggest that morphology is largely derivable from the word form alone but context can improve the picture.

Subword tokenization does not work well with small models. Subword models regardless of the usage of context, are over 5% point worse than the non-subword models despite using over 100 times more parameters. We attribute this to data sparsity. The input is tokenized with mBERT's tokenizer but most subwords never appear in the 2000 training examples of a particular probing task. Still, the input vocabulary tends to be very large for such a small dataset. swLSTM models use 5600 subwords-per-task on average, while swLSTM-t models use 1625 subwords. This means that the target word form uses 1625 different subwords, most of them occurring only once. We performed the same set of experiments with XLM-RoBERTa's tokenizer and the results are very similar.

XLM-large and XLM-MLM. XLM-large is the best model, it slightly outperforms its smaller (a.k.a. *base*) variant, XLM-RoBERTa and XLM-MLM. However, this comes at a cost of a much larger model size[7]. XLM-MLM is also twice as large as XLM-RoBERTa and their performance is very similar. We added XLM-large and XLM-MLM for the sake of completeness but since mBERT does not have a large variant, the base XLM-RoBERTa model is more comparable to mBERT and we focus our main analysis on the base model.

DistilmBERT. Distillation clearly comes at a cost, the overall accuracy is reduced by 2.7% points. This means a 28% increase in error rate. DistilmBERT is still better than any of the baselines and the merits of distillation on cost efficiency are undeniable in higher level applications.

mT5. mT5 is a generative model for 100 languages and as such we expect lower performance when using only its encoder. The results show that it is in fact much worse for this task than most of our baselines.

[7]The batch size had to be reduced for these experiments

Tag	POS	mBERT	XLM-R	XLM-large	XLM-MLM	Stanza*	GPT-4o	GPT-4o-mini
case	adj	87.6	90.5	90.0	89.9	93.3	84.6	62.4
	noun	88.8	91.4	90.8	90.8	90.6	83.6	65.7
	propn	87.5	89.3	88.2	89.0	89.1	79.8	65.5
	verb	88.5	90.9	91.3	92.5	90.1	74.7	45.6
gender	adj	87.6	86.3	87.2	88.8	91.9	92.3	82.0
	noun	87.2	88.6	90.4	88.1	90.0	93.0	84.9
	propn	75.6	76.5	77.6	76.8	70.4	79.0	77.3
	verb	88.0	88.5	88.9	88.6	86.7	87.3	71.5
number	adj	96.4	97.1	97.3	97.2	96.3	96.0	84.5
	noun	93.5	95.1	95.1	94.4	94.2	95.9	91.2
	propn	89.2	90.1	92.3	90.4	78.7	91.0	73.0
	verb	95.8	97.0	97.4	96.3	95.7	97.3	90.3
tense	adj	98.0	99.5	100.0	100.0	98.0	89.5	87.5
	verb	91.7	94.1	95.5	92.0	91.1	91.1	88.0
ALL		90.4	91.8	92.2	91.5	91.4	90.9	80.8

Table 6.13: Average test accuracy over all languages by task and model. We excluded the weakest (cf. Figure 6.19) models, mT5 and DistilmBERT, for brevity. The last row is the average of all 247 tasks. The Stanza results do not include the 6 Albanian tasks due to the lack of Albanian models.

Tag	POS	mBERT	XLM-R	chLSTM	swLSTM	chLSTM-t	swLSTM-t	fastText
case	adj	87.6	90.5	77.8	74.5	69.1	61.4	70.3
	noun	88.8	91.4	78.8	74.4	70.1	61.6	73.7
	propn	87.5	89.3	78.9	76.8	67.4	60.8	68.4
	verb	88.5	90.9	80.2	75.1	79.0	67.4	78.9
gender	adj	87.6	86.3	83.5	76.2	78.1	70.3	80.3
	noun	87.2	88.6	78.9	72.6	77.3	67.8	80.9
	propn	75.6	76.5	64.4	60.0	62.0	54.1	69.0
	verb	88.0	88.5	87.8	83.4	86.9	82.5	87.0
number	adj	96.4	97.1	91.8	87.4	89.8	83.2	89.4
	noun	93.5	95.1	88.9	83.7	87.3	80.4	90.2
	propn	89.2	90.1	83.3	77.2	80.4	73.9	78.4
	verb	95.8	97.0	95.1	91.0	93.9	89.2	95.2
tense	adj	98.0	99.5	100.0	98.1	98.8	97.6	96.6
	verb	91.7	94.1	90.7	83.7	89.3	81.0	91.0
ALL		90.4	91.8	85.0	79.9	81.5	74.2	83.6

Table 6.14: Average test accuracy over all languages by task and model including all baselines. The last row is the average of all 247 tasks. The prefix *ch* means that the model uses character tokenization, while *sw* means it uses subword tokenization. The postfix *-t* means that the model only uses the target word form and ignores its context.

GPT-4o performs the same as PLMs on average. The overall accuracy of GPT-4o is between the accuracy of mBERT and XLM-RoBERTa but this is partially due to GPT-4o struggling with some tasks. If we disregard the bottom 20% of tasks by GPT-4o's accuracy, its average accuracy jumps to 95.1% while mBERT and XLM-RoBERTa are only 92.5% and 93.8% correct on the same set of tasks. We analyze this in more detail in Section 6.5.6.

GPT-4o-mini is suboptimal for this task. GPT-4o-mini is 10% point worse than GPT-4o and it also underperforms both chLSTM and target only chLSTM (cf. Table 6.14).

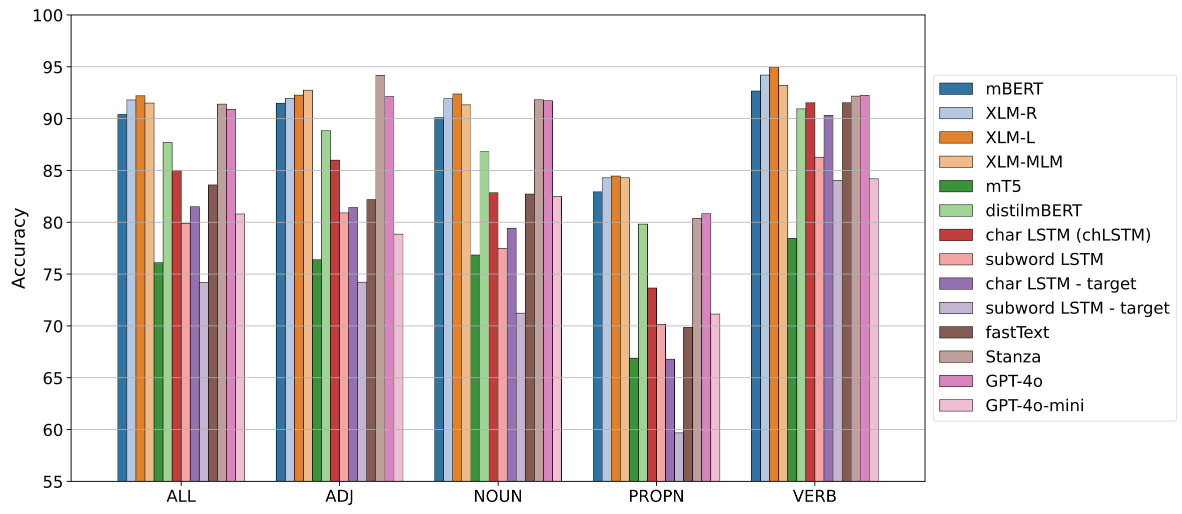

Figure 6.19: Accuracy of the pre-trained and the baseline models grouped by POS.

6.5.3 mBERT and XLM-RoBERTa against chLSTM

In this section we analyze the general morphological capabilities of mBERT and XLM-RoBERTa in comparison with the strongest baseline, chLSTM. We do this way because many of the morphological tasks are easy to begin with and chLSTM, a baseline trained solely on the probing data, is already well over 90%. It is only when PLMs are noticeably better than chLSTM, that we can assume that morphosyntactic information is in fact stored somewhere in the pre-trained weights.

Tables 6.13 and 6.14 list the accuracy of each task averaged over the languages the task is available in. Perhaps the most salient fact about these results is that PLMs perform over 85% in each task except ⟨PROPN, Gender⟩. Proper nouns are the most difficult regardless of the morphological tag. Pre-trained models reduce the error rate of the best baseline by 50% in spite of the relatively low (40k) number of trainable parameters. The baselines have a similar or higher number of trainable parameters which suggests that some of the morphosyntactic information must come from the pre-trained weights of mBERT and XLM-RoBERTa.

Figure 6.20 shows a task-by-task comparison color coded by tag. It is evident that the large majority of tasks are over the diagonal, in other words, the pre-trained models are better. There are 8 exceptions in each model comparison but only 4 gender tasks (and ⟨Estonian, ADJ, Tense⟩ for mBERT) are where the difference is statistically significant. Number and tense easy for all three models, while gender is much better retrieved from PLMs than from chLSTM.

Figures 6.21 and 6.22 show the difference between the accuracy of PLMs and chLSTM averaged over language families. chLSTM is only better than one or both of PLMs in 8 tasks out of the 247 and the difference is never large.

We find a large number of tasks at the other end of the scale. Particularly Slavic case and gender probes work much better in both mBERT and XLM-RoBERTa than in chLSTM. Slavic languages have highly complex declension with 3 genders, 6–8 cases, and frequent syncretism. This explains why chLSTM is struggling to pick up the pattern from 2,000 training samples alone. mBERT and XLM-RoBERTa were both trained on large datasets in each language and therefore may have picked up a general representation of gender and case.[8] It is also worth mentioning that among the 100 lan-

[8]Slavic conjugation is not much simpler, but concentrates on the word not on the context; see Chapter 8.

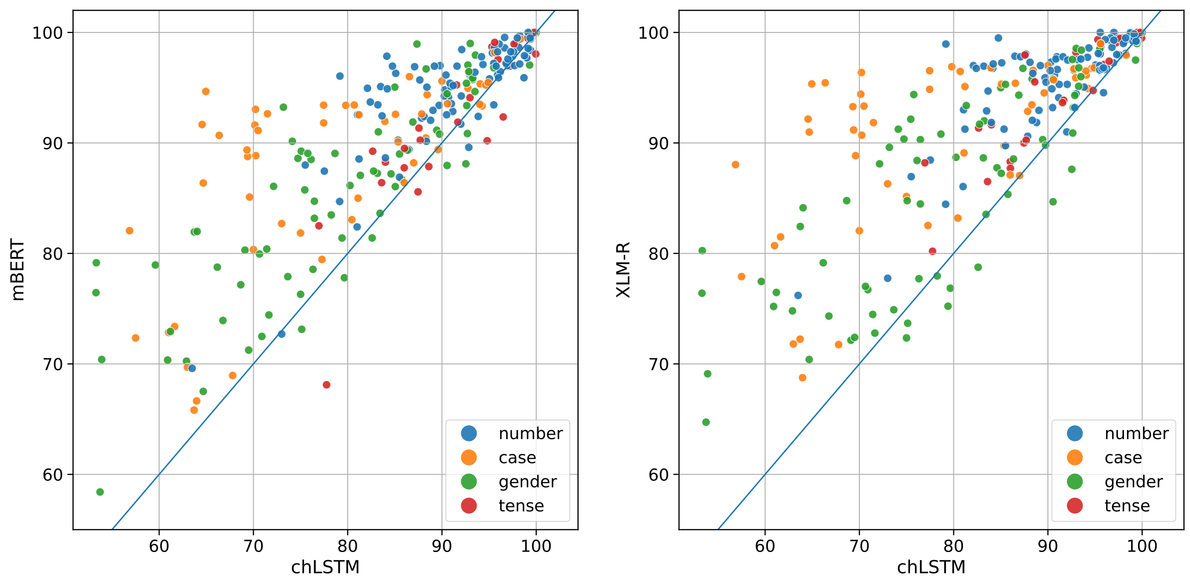

Figure 6.20: chLSTM's accuracy on each probing task in comparison with mBERT and XLM-RoBERTa. We added a $x = y$ diagonal for clarity.

guages that these models support, Slavic languages are one of the largest language families with 10 or more languages. They are also highly overrepresented in UD and hence in our probing dataset (cf. Section 6.3). Figure 6.23 shows the differences for each Slavic language and task. The similarities appear more geographical (Ukrainian and Belarus, Czech and Polish) than historical, though the major division into Eastern, Western, and Southern Slavic is still somewhat perceptible.

We next examine the languages that PLMs tend to struggle with and the ones that they handle with ease. Figure 6.24 shows the bottom five and the top five languages by the average accuracy of PLMs on tasks in these languages. Irish, Latin, Icelandic, Albanian and German are the languages that the models struggle with the most, although the results widely differ task-by-task as measured by the large standard deviation. We find Catalan, English, Armenian, French and Turkish at the other end of the scale. Since chLSTM is also very good at these languages, we conclude that our morphosyntactic probing tasks are simply easy to solve with 2000 training examples. French is a possible exception but this is mainly caused by chLSTM's low performance on gender tasks.

6.5.4 Comparison between mBERT and XLM-RoBERTa

Table 6.13 showed that XLM-RoBERTa is slightly better than mBERT on average and in every POS-tag category except ⟨ADJ, Gender⟩. However, this advantage is not uniform over tag and POS as evidenced by Figure 6.25, which shows the number of tasks where one model is significantly better than the other. XLM-RoBERTa is always better or no worse than mBERT at case and tense tasks with the exception of ⟨Swedish, NOUN, Case⟩ and ⟨Romanian, VERB, Tense⟩, where mBERT is the stronger model.

Figure 6.26 illustrates the same task counts by language family. We observe the same performance in most tasks from the Germanic and Romance language families. XLM-RoBERTa is better at the majority of the tasks from the Semitic, Slavic, and Uralic families, and the rest are more even. Interestingly the two members of the Indic family in our dataset, Hindi and Urdu behave differently. XLM-RoBERTa is better at 5 out of 6 Hindi tasks and the models are the same at the sixth task. mBERT on the other

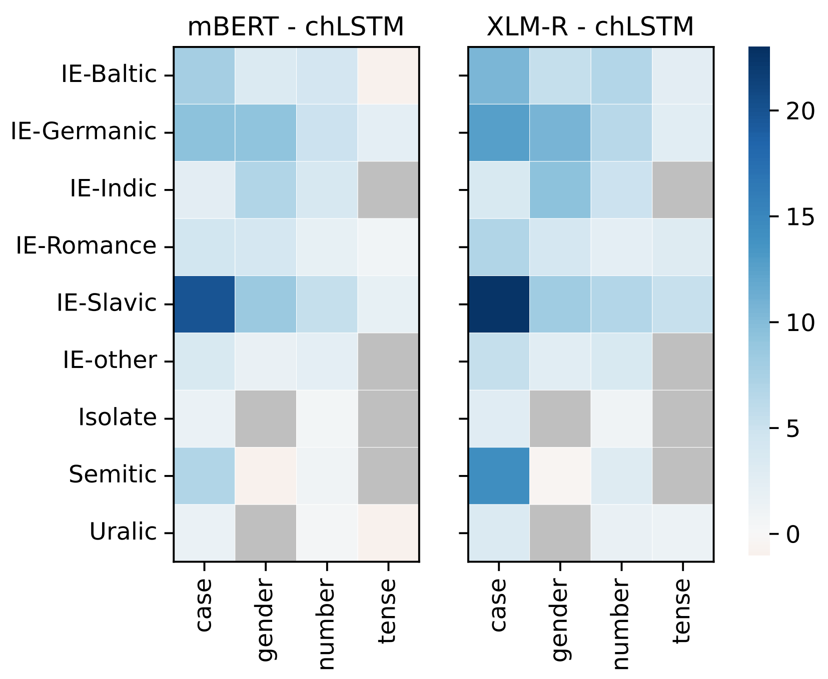

Figure 6.21: Difference in accuracy between mBERT (left) and chLSTM, and XLM-RoBERTa (right) and chLSTM grouped by language family and morphological category. Grey cells represent missing tasks.

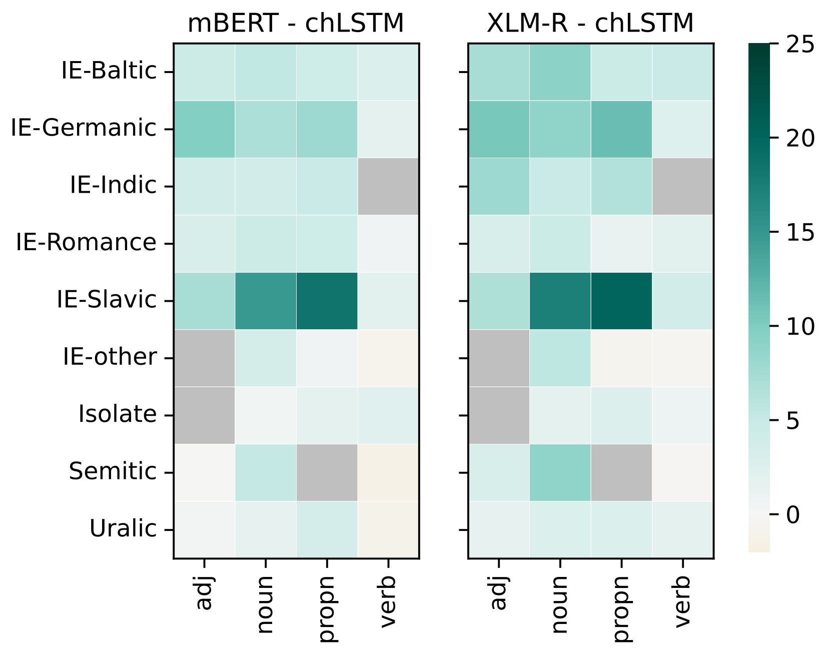

Figure 6.22: Difference in accuracy between mBERT (left) and chLSTM, and XLM-RoBERTa (right) and chLSTM grouped by language family and POS. Grey cells represent missing tasks.

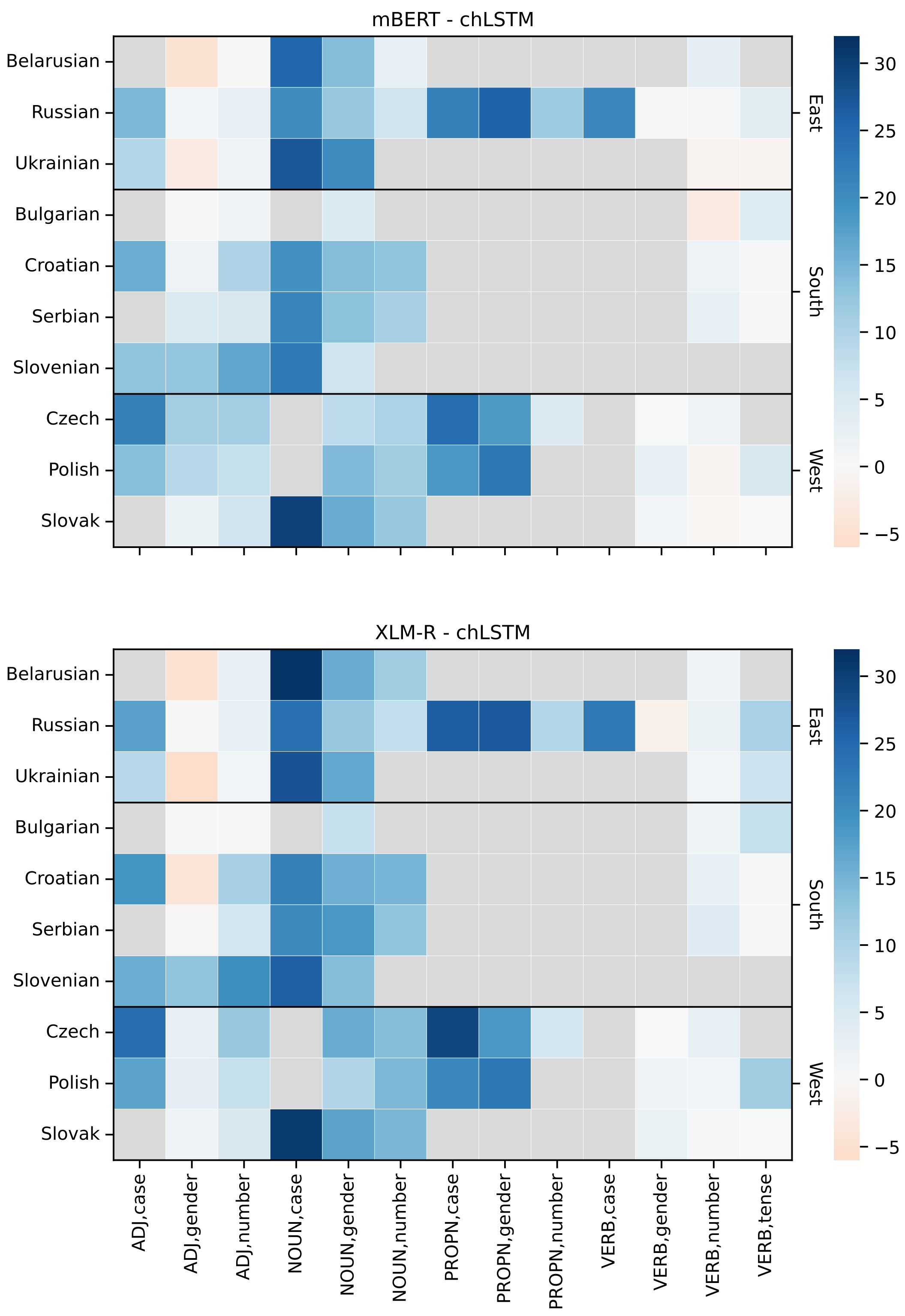

Figure 6.23: Task-by-task difference between PLMs and chLSTM in Slavic languages. Grey cells represent missing tasks.

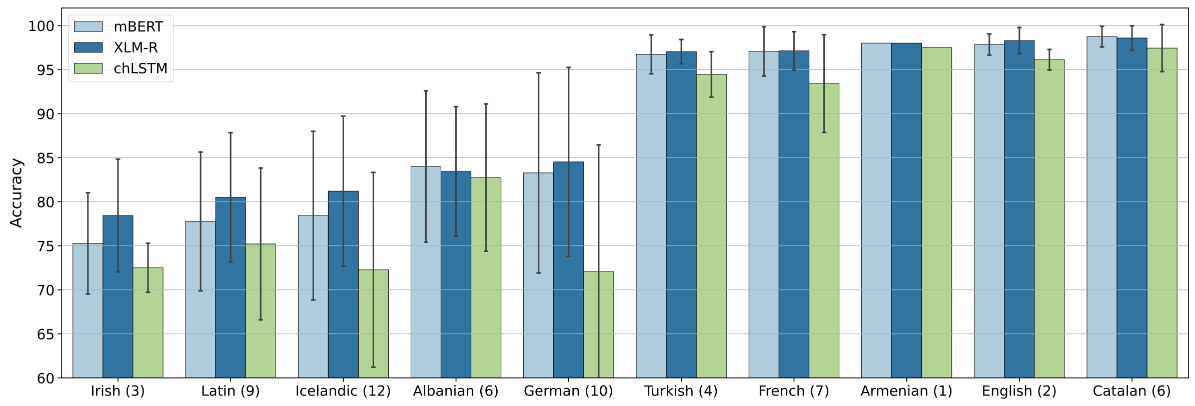

Figure 6.24: The best and the worst 5 languages by the average performance of mBERT and XLM-RoBERTa. Thin lines represent the standard deviation. The number of tasks in a particular language is listed in parentheses.

hand, is better at one Urdu task and the models are the same at other three Urdu tasks. This might be due to the subtle differences in mBERT and XLM-RoBERTa subword tokenization (cf. Section 5.3.1).

6.5.5 Difficult tasks

Some morphosyntactic tags are hard to retrieve from the model representations. In this section we examine such tags and the results in more detail. Table 6.15 lists the 10 hardest tasks measured by the average accuracy of mBERT and XLM-RoBERTa. ⟨German, PROPN, Case⟩ is difficult for two reasons. First, nouns are not inflected in German; case is marked in the article of the noun. The article depends on both the case and the gender and syncretism (ambiguity) is very high. This is reflected in the modest results for ⟨German, NOUN, Case⟩ as well (72.9% for mBERT, 80.7% for XLM-RoBERTa). Second, proper nouns are often multiword expressions. Since all tokens of a multiword proper noun are tagged PROPN in UD, our sampling method may pick any of those tokens as a target token of a probing task.

Another outlier is ⟨Arabic, ADJ, Case⟩. Arabic adjectives usually follow the noun they agree with in case. With the elative there is no agreement, and sometimes the adjective precedes the noun which is in genitive, but the adjective is not. This kind of exceptionality may simply be too much to learn based on relatively few examples – it is still fair to say that grammarians (humans) are better pattern recognizers than PLMs.

Although the difficult tasks for mBERT and XLM-RoBERTa tend to be the same, this is not the case for GPT-4o, which struggles with some of the tasks from Table 6.15 and excels with others such as ⟨Latin, ADJ, case⟩. We analyze this in more detail in the next section.

6.5.6 GPT-4o

The results of GPT-4o prompting fall between XLM-RoBERTa and mBERT on average but the standard deviation over the 247 tasks is higher (10.5 vs. 8.2). Figure 6.27 shows a task-by-task comparison between GPT-4o and the PLMs. GPT-4o is clearly better at gender tasks and worse at case tasks than mBERT and XLM-RoBERTa. The biggest gap between GPT-4o and mBERT (or XLM-RoBERTa) is in case tasks in Hindi, Urdu, Finnish and Basque. Case in agglutinative languages is usually marked at the word level hence simple orthographic cues are very useful. chLSTM, which relies on orthography

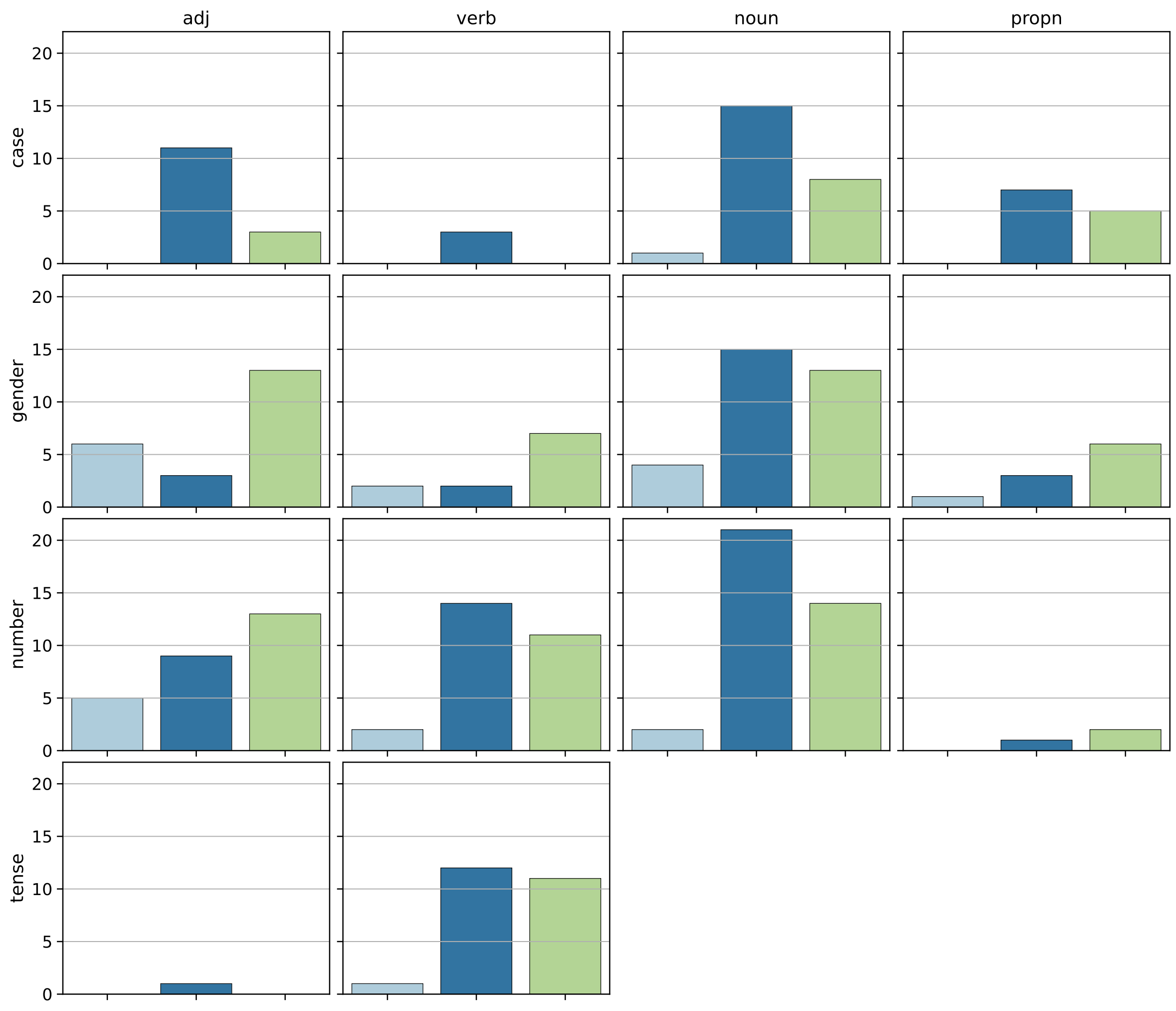

Figure 6.25: mBERT XLM-RoBERTa comparison by tag and by POS.

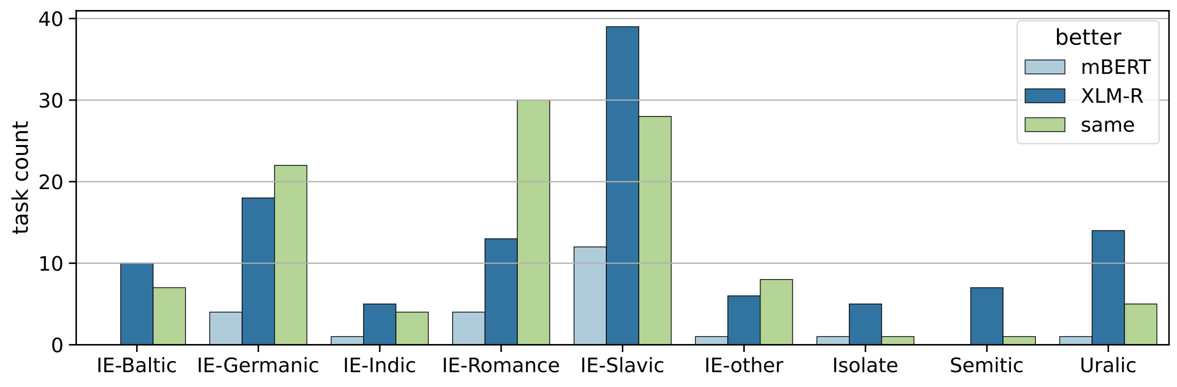

Figure 6.26: mBERT XLM-RoBERTa comparison by language family.

task	mBERT	XLM-R	chLSTM	Stanza	GPT-4o
⟨Icelandic, PROPN, gender⟩	58.4	64.7	53.7	59.2	79.1
⟨German, PROPN, case⟩	66.6	68.8	64.0	71.5	71.0
⟨Icelandic, ADJ, gender⟩	67.5	70.4	64.7	82.1	83.1
⟨Arabic, ADJ, case⟩	65.8	72.2	63.7	81.6	67.2
⟨German, PROPN, gender⟩	70.4	69.1	53.9	78.6	74.1
⟨Albanian, NOUN, case⟩	69.0	71.8	67.8		55.5
⟨Latin, ADJ, case⟩	69.7	71.8	63.0	81.0	90.5
⟨Irish, NOUN, gender⟩	71.2	72.4	69.5	80.0	80.5
⟨Dutch, PROPN, gender⟩	70.2	74.8	62.9	65.0	53.0
⟨Hindi, PROPN, gender⟩	70.4	75.2	60.9	68.0	77.5

Table 6.15: 10 hardest tasks for PLMs.

alone, performs much better than GPT-4o. On the other hand gender tasks, where GPT-4o excels, often require lexical knowledge of the language. Perhaps this is the tradeoff we pay for GPT-4o's high level capabilities.

The results of GPT-4o are clearly very different from that of the mBERT and XLM-RoBERTa. The two smaller models behave very similarly evidenced by their high correlation over the 247 tasks (0.95 Person correlation). GPT-4o on the other hand has a more modest correlation, 0.55 with either models. GPT-4o has some wild outliers where its performance is well below the baselines. The picture is very similar when we use Spearman's rank correlation to counteract outliers. Although the gap is smaller (0.94 between mBERT and XLM-RoBERTa and 0.69–0.7 between GPT-4o and the other two models) but the difference in model behavior is obvious. It may be beneficial to tweak the prompt but it falls outside of the scope of this thesis.

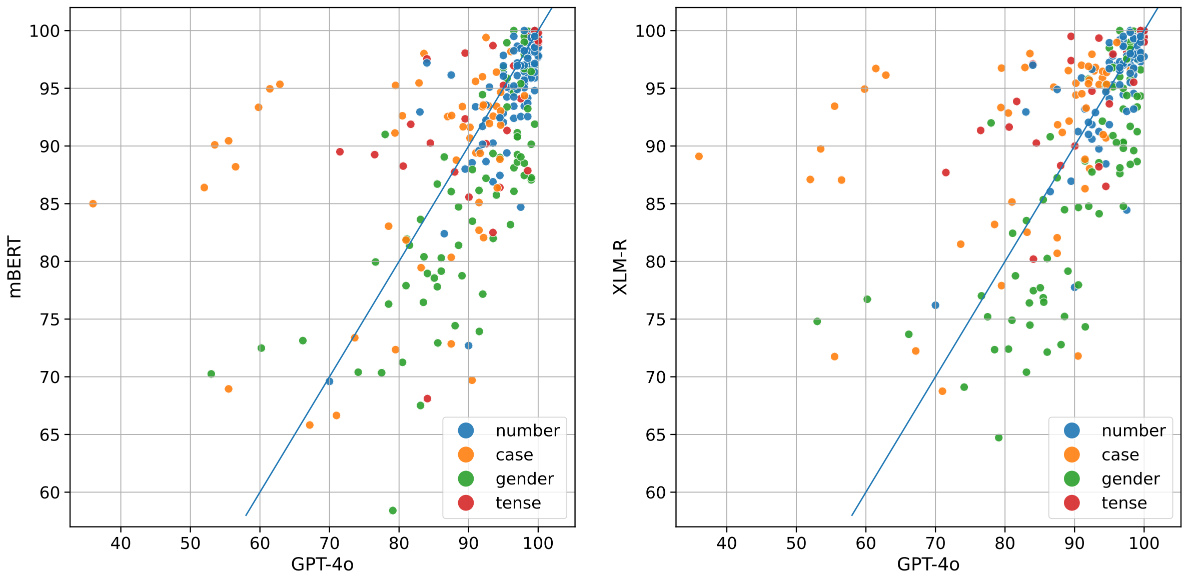

Figure 6.27: GPT-4o's accuracy on each probing task in comparison with mBERT and XLM-RoBERTa. We added a $x = y$ diagonal for clarity.

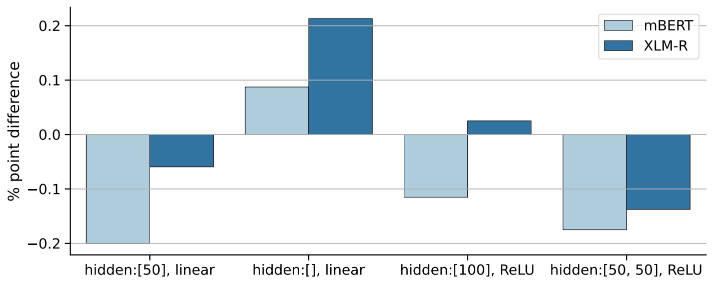

Figure 6.28: The average probing accuracy using different MLP variations. We indicate the size(s) of hidden layer(s) in square brackets.

6.6 Ablations

In this section we empirically consider the criticisms raised in Belinkov (2021) and Ravichander et al. (2021) for probing setups like ours. Our first group of tests (6.6.1) confirms that the probing accuracy does not depend on the choice of probe, in particular, linear probes are no better or worse than non-linear ones. We also show that probing individual layers of mBERT or XLM-RoBERTa is worse or no better than probing the weighted sum of all layers (6.6.2). We also show that fine-tuning decreases the probing accuracy while it substantially increases the computational requirements (6.6.3). Finally, we show that probing a randomly initialized model, a control used by Voita and Titov (2020), is significantly worse than probing the trained model (6.6.4). We present all results averaged over all 247 tasks in this section. With the exception of 6.6.4, we do not perturb the input sentences. Finally, we perform a few ablations specific to BERT models in Section 6.6.6.

6.6.1 Linear probing and MLP variations

The probes we presented so far all use an MLP with a single hidden layer with 50 neurons. The input is the weighted sum of the 12 layers and the embedding layer with learned weights. The size of the output layer depends on the number of classes in the probing tasks. We use ReLU activations in the MLP.

The original BERT paper used a simple linear classification layer with weight $W \in \mathbb{R}^{K \times H}$, where K is the number of labels and H is the hidden size of BERT, 768 in the case of mBERT and XLM-RoBERTa. Hewitt and Liang (2019) argue that linear probes have high selectivity, i.e., they tend to memorize less than non-linear probes. We test this on our dataset with two kinds of linear probes. The first one is the same as the general probing setup but we remove the ReLU activation. The second one completely removes the hidden layer similarly to the original BERT paper. We also test two MLP variations, one with 100 hidden size instead of 50 and another one with two hidden layers.

Figure 6.28 shows the accuracy of the two linear probes and the larger MLPs as the difference from the default version we used elsewhere. These numbers are averaged over that 247 tasks. The differences are all smaller than 0.25% points. These results indicate that the probing accuracy does not depend on the probe type and particularly that linear probes perform similarly to non-linear ones.

6.6.2 Layer pooling

Our default setup uses a weighted sum of the 12 layers and the embedding layers, one scalar weight for each of them with a total of 13 learned weights. It has been shown (Tenney et al., 2019a) that the different layers of mBERT work better for different tasks. Lower layers work better for low level tasks such as POS tagging, while higher layers work better for higher level tasks such as coreference resolution. Morphosyntactic tagging is a low level task. The embedding layer itself is often indicative of the morphological role of a token. We test this by probing each layer separately as well as probing the concatenation of all layers.

Figure 6.29 shows the difference between probing the weighted sum of all layers and probing individual layers averaged over all tasks. We observe approximately 10% point difference in the embedding layer (layer 0) and the lower layers. This difference gradually decreases and it is close to 0 in the upper layers. Our results support the finding of Hewitt et al. (2021) that morphosyntactic cues are encoded much higher into the layers[9] then previously suggested by Tenney et al. (2019a), discussed in Hewitt and Liang (2019). Layer concatenation (concat) is slightly better than the weighted sum of the layers, but it should be noted that the parameter count of the MLP is an order of magnitude larger thanks to the 13 times larger input dimension.

We also observe that the gap between the embedding layer (layer 0) and the first Transformer layer (layer 1) is much smaller in the case of XLM-RoBERTa than mBERT. XLM-RoBERTa's embedding layer is significantly better than mBERT's embedding layer to begin with (82.2% vs. 80%) and this gap shrinks to 0.3% point at the first layer (82.2% and 81.9%). This is more evidence for one of our main observations that XLM-RoBERTa's embedding and vocabulary are better than those of mBERT's.

We further examine the layer weights in our 'default setup' where the layers are weighted by 13 scalar weights. The 13 scalar weights are learned for each probing experiment along with the MLP weights. We use the softmax function to produce a probability distribution for the layer weights. Our analysis shows that the layer distribution is very close to the uniform distribution. We quantify this in two ways: the ratio of the highest and the lowest layer weight, and the entropy of the weight distribution. The two measures highly correlate (0.95 Pearson correlation). The highest ratio of the largest and smallest layer weight is 2.22 for mBERT and 2.24 for XLM-RoBERTa, while the lowest entropy is 3.62 for both models (the entropy of the uniform distribution is 3.7). We list the 50 task with the highest max-min ratio in Figure 6.30. Interestingly the higher layers are weighted higher in each of the 50 outlier tasks again contradicting the notion that low level tasks such as morphology use lower layers of BERT (Tenney et al., 2019a).

We also found that the two models show similar patterns task-wise. When the layer weights corresponding to one task have a lower entropy, they tend to have a lower entropy when we probe the other model too. The ranking of the entropy of the 247 tasks' weights confirms this (0.63 Spearman's rank correlation). Most of the outlier tasks are from the Slavic family or are in German. This is similar to what we learn from the Shapley outlier analysis later (Section 8.2.3).

6.6.3 Fine-tuning

Fine-tuning (as opposed to feature extraction), trains all BERT parameters along with the MLP in an end-to-end fashion. This raises the number of trainable parameters from 40k to over 170M. The recommended optimizer for fine-tuning BERT models is AdamW (Loshchilov and Hutter, 2019), a variant of the Adam optimizer. We try both Adam and AdamW for fine-tuning the model on each task and show that AdamW is indeed a better choice than Adam (Table 6.16) but nevertheless the feature

[9]This holds for not just XLM-RoBERTa, but mBERT as well.

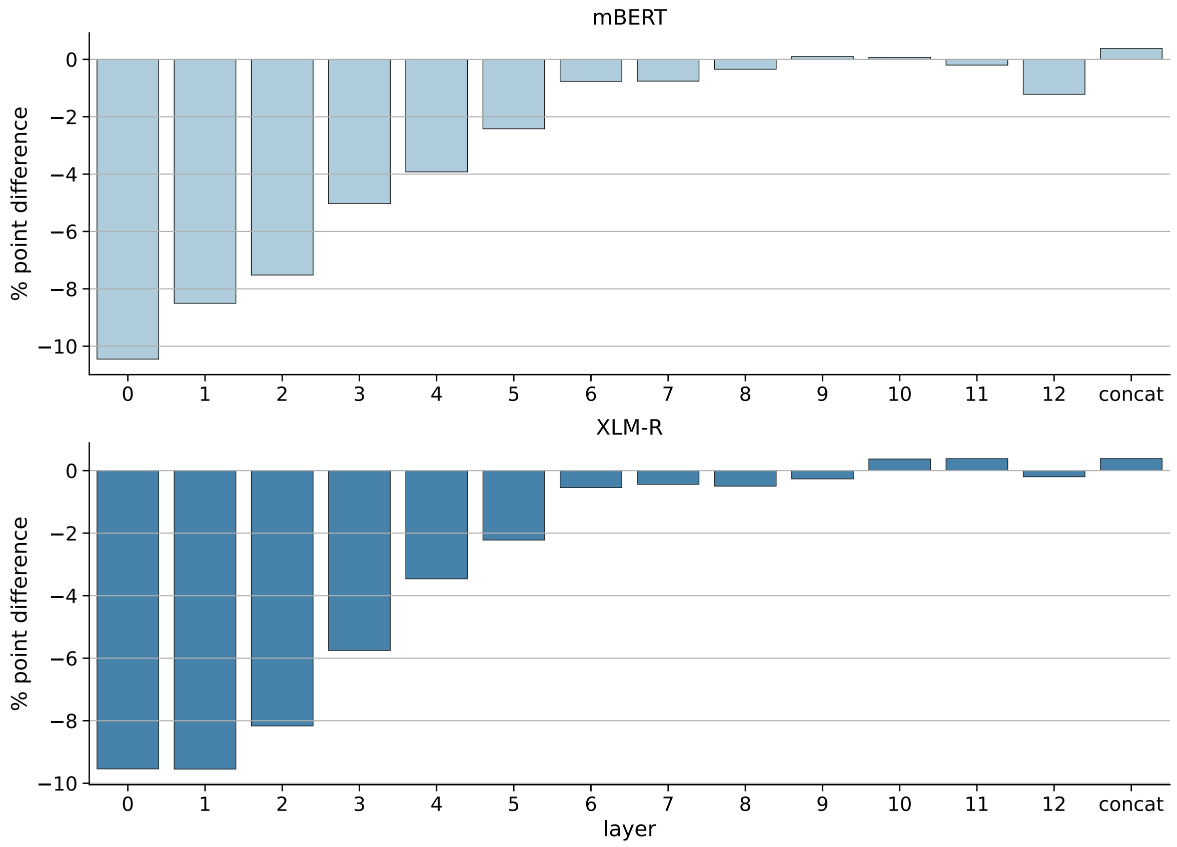

Figure 6.29: The difference between probing a single layer and probing the weighted sum of layers. *concat* is the concatenation of all layers. *0* is the embedding layer. Large negative values on the y-axis mean that probing the particular layer on the x-axis is much worse than probing the weighted sum of all layers.

	Adam	AdamW
Fine-tuning	76.2	89.2
Feature extraction	90.4	89.9

Table 6.16: Comparison of fine-tuned and frozen (feature extraction) models.

extraction results are 0.5 percentage point better than the fine-tuning results and this difference is statistically significant. Our experiments also show that the running time is increased 80-fold when we fine-tune mBERT. Due to this increase in computation time, we do not repeat the experiments for XLM-RoBERTa. It should be noted that BERT fine-tuning has its own tricks (Houlsby et al., 2019; Li and Liang, 2021; Ben Zaken et al., 2022) that may lead to better results but we do not explore them in this thesis.

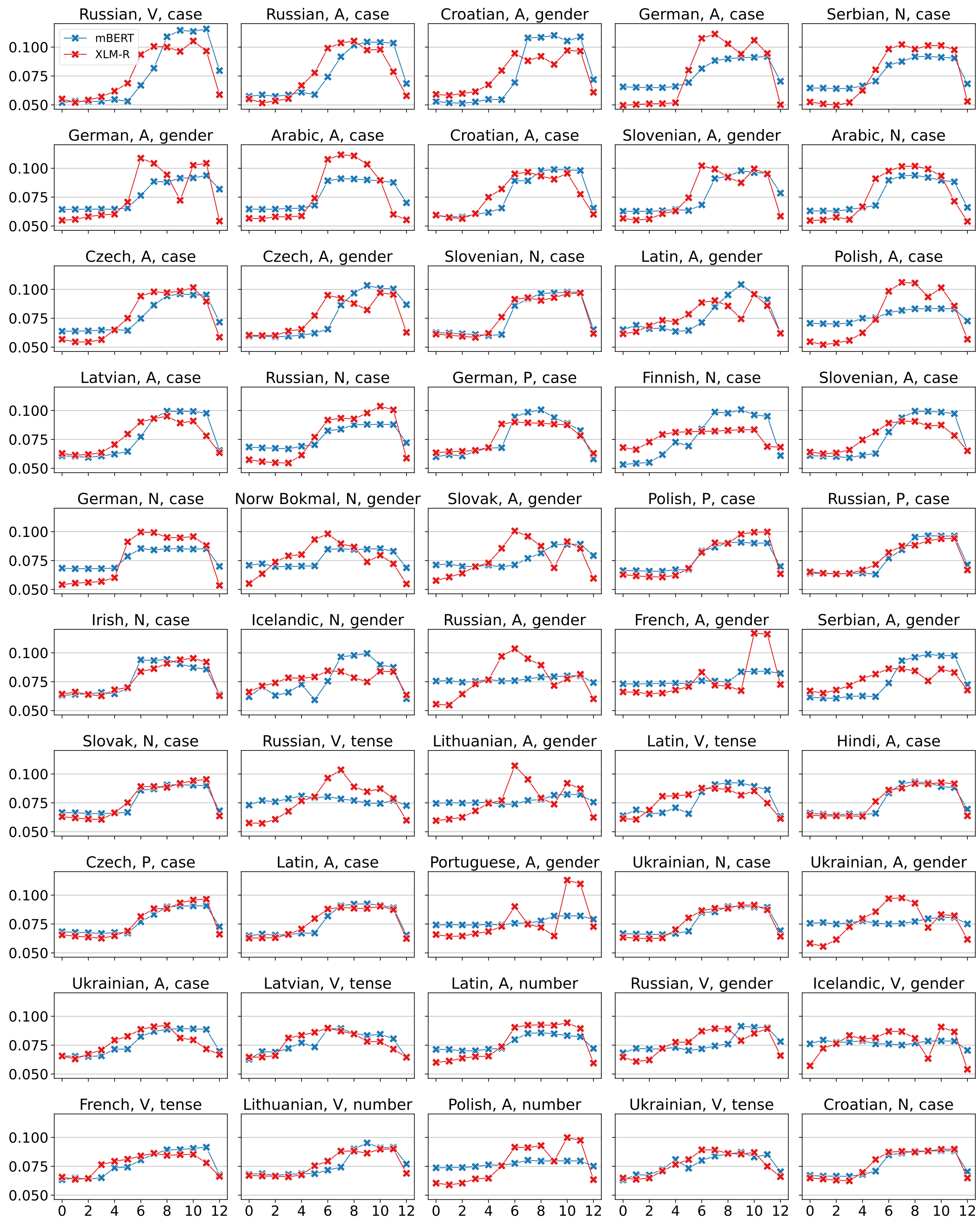

Figure 6.30: Layer weight outliers. Layer 0 is the embedding layer.

	Pre-trained	Random layers	Fully random	Majority baseline
mBERT	90.4	73.5	67.0	41.0
XLM-RoBERTa	91.8	74.7	70.1	41.0

Table 6.17: Probing accuracy on the randomly initialized mBERT and XLM-RoBERTa models.

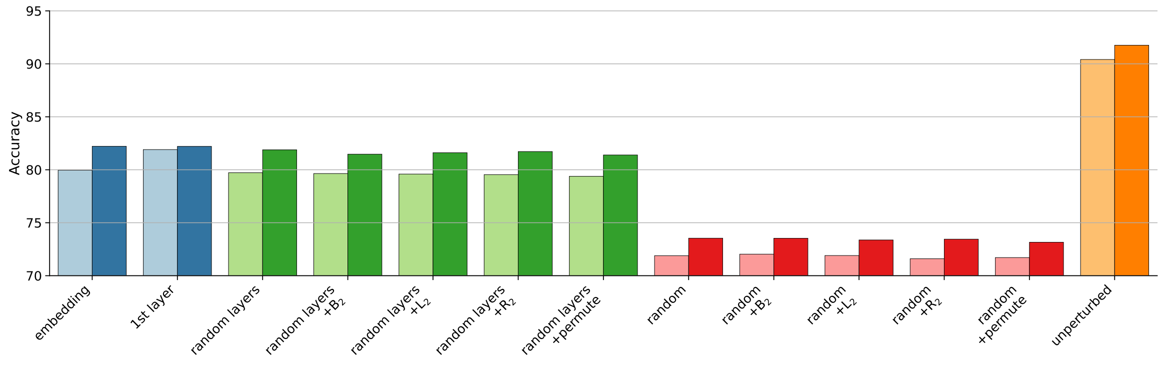

Figure 6.31: Random mBERT and random XLM-RoBERTa performance comparison with different perturbation setups and the unperturbed trained model variants (orange bars). Left-to-right: Blue: Accuracy of the embedding and first layers' probes; Green: Random models with pre-trained embedding layer: no perturbation, B2, L2, R2, PERMUTE; Red: Random models where the embedding layer is random as well: no perturbation, B2, L2, R2, PERMUTE; Orange: Unperturbed trained models. Perturbations are introduced in Chapter 8.

6.6.4 Randomly initialized PLMs

Randomly initialized language models have been widely used as a baseline when evaluating language models (Conneau et al., 2017), especially via auxiliary classifiers (Conneau et al., 2018b; Zhang et al., 2018; Htut et al., 2019; Voita and Titov, 2020). Zhang et al. (2018) showed that the mechanism of assigning morphosyntactic tags to these random embeddings is significantly different. As they demonstrated, randomly initialized PLMs rely on word identities, while their trained counterparts maintain more abstract representations of the tokens in the input layer. Therefore, probing classifiers applied on random PLMs may pick up low-level patterns only, such as word identity, and this could mislead the probing controls when used as a baseline.

In order to test this hypothesis, we trained the probing classifiers using randomly initialized mBERT and XLM-RoBERTa models. In this setup both fully random and pre-trained embedding layer with random Transformer layers were compared to trained PLMs.

Table 6.17 shows the overall probing accuracy achieved on random models. We add the majority (most frequent) baseline as a comparison. Although the random PLMs are clearly better than the majority baseline, they are far worse than the trained PLMs.

We tested the perturbations introduced in Chapter 8 in combination with random models as well.

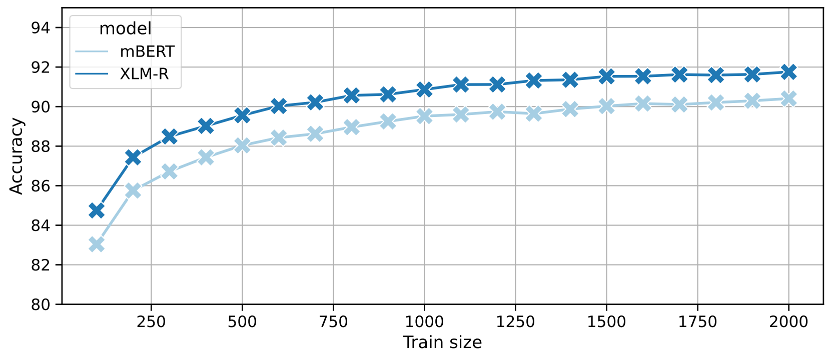

Figure 6.32: Probing accuracy with reduced training data.

Figure 6.31 shows that neither B2, L2, R2 nor PERMUTE[10] perturbations affect the model's performance in a way that they do when applied on the trained models (compare Figure 8.1). Nevertheless, this sub-experiment offers two further supporting arguments for the claims Zhang et al. (2018) made about random PLMs learning word-identities:

1. The accuracies of random models' morphological probes match the accuracies of their embedding layers' probe; i.e. even the Transformer-based random PLMs rely *mostly* on the word-identities represented by their embeddings

2. Probing perturbed and unperturbed embeddings of random PLMs does not make a big difference (the accuracies of unperturbed and perturbed models' are less than 1% apart). Clearly, word identities count the most, their order almost not at all.

Based on this finding we do not use randomized PLMs as baselines.

6.6.5 Training data size

Our sampling method (cf. 6.2.2) generates 2,000 training samples. Raising this number would remove many tasks from languages with smaller UD treebanks. Probing methods on the other hand are supposed to test the already existing linguistic knowledge in the model, therefore probing tasks' training sets should not be too large. In this section, we show that smaller training sizes result in inferior probing accuracy. Our choice of 2,000 samples was a practical upper limit that allowed for a large number of tasks from mid-to-large sized UD treebanks.

Figure 6.32 shows the average accuracy of the probing tasks when we use fewer training samples. Although the probing tasks work considerably better than the majority baseline even with 100 sentences, the overall accuracy gets better as we increase the training data. Interestingly, XLM-RoBERTa is always slightly better than mBERT.

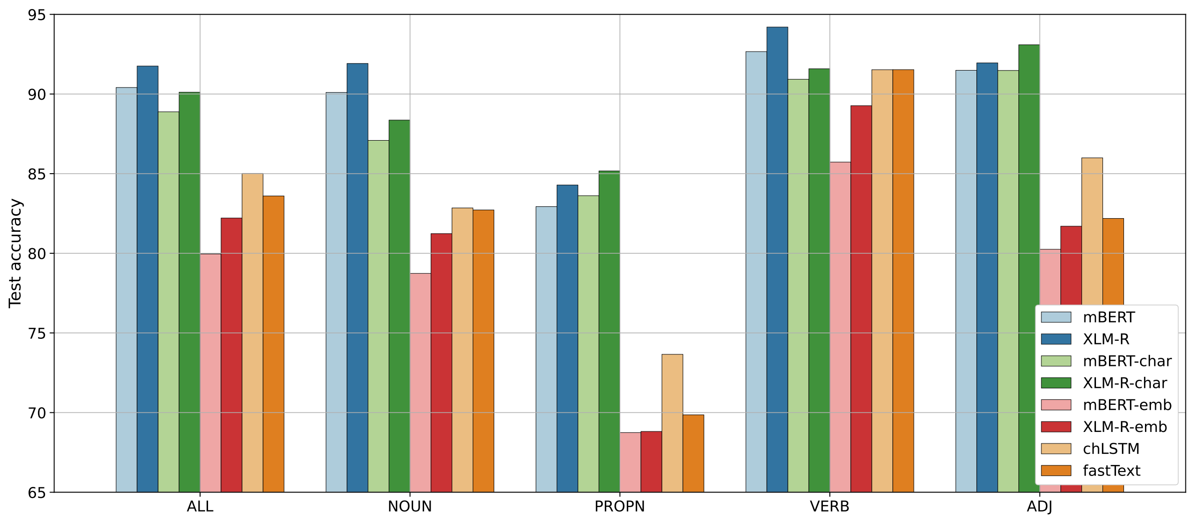

Figure 6.33: The performance of ablated BERT models grouped by POS.

6.6.6 Model ablations

In this section we focus on ablations specific to the BERT model architecture and its subword tokenizer. We perform three types of ablations.

char removes the tokenization aspect by replacing the subword tokenization with character tokenization. Since the subword vocabularies of both mBERT and XLM-RoBERTa contain all individual characters, we can artificially simulate character tokenization. For example the normal mBERT tokenization of the word *apple* is `[app + ##le]` but we change this into `[a + ##p + ##p + ##l + ##e]`. The XLM-RoBERTa tokenization of apple changes from `[_apple]` to `[_a + p + p + l + e]`[11]. Using character tokenization instead of subword tokenization substantially increases the sequence length so much so that it sometimes exceeds the maximum input length (512) supported by PLMs. To avoid this limitation, we only use character tokenization on the target word form and use subword tokenization on the rest of the sentence. As Figure 6.33 shows, the overall effect is less than 5% point but it varies by POS. Nouns and verbs are affected the highest while pronouns and adjectives seem to have a small but positive effect. This suggests that the left-ward greedy tokenization of subword tokenizers is often suboptimal from a morphosyntactic viewpoint.

emb uses the output of the subword embedding layer alone and ignores the 12 Transformer layers. The embedding itself is a static 2D matrix which assigns a 768 dimensional dense vector to each subword in the subword vocabulary. This also means that the context is only incorporated at higher layers. This ablation causes a 15 to 20% drop in all POS categories except verbs. Verbal morphology is most often expressed on the word form itself, while adjectives have agreements with subject or the object of the sentence, so it is unsurprising that we see a much larger gap in adjectives than in verbs. EMB is similar to fastText in that they are both static embeddings trained on sentences but fastText is

[10] L_2 replaces two tokens before the target token with a mask symbol, R_2 replaces two tokens after and B_2 replaces two tokens on both sides. PERMUTE shuffles the word order of the sentence. See Section 8.1 for the motivation of these perturbations and examples.

[11] The Unicode lower eights block is replaced with an underscore in the example

better in every POS category than mBERT and XLM-RoBERTa used as *traditional* embeddings. emb is the same as pooling the 0th layer in Section 6.6.2.

nocontext removes everything from the sentence except the target word but unlike emb it uses the embedding and all Transformer layers. This is our first test on the importance of the context for morphosyntactic probing. We greatly extend this study in Chapter 8. nocontext is fairly similar to emb except for verbs, where nocontext is a better method.

Our model ablations show that PLMs rely on multiple components and removing one results in significant loss in performance. They do not work well as static embeddings or without the sentence context in their input. Disabling the subword tokenization also has a moderate negative effect on morphosyntactic probing.

Chapter 7

Language-specific Models

In Chapter 5 we introduced the BERT model family with its initial English and multilingual checkpoints. The first non-English monolingual models[1] appeared a few months after BERT's release. Within the next year most high resource languages had at least one language-specific BERT model and the list was growing every day. The first Uralic-language BERT model was FinBERT (Virtanen et al., 2019), a Finnish model.

In this chapter, we extend our wide-scale evaluation efforts from Chapter 6 to monolingual models, particularly models for Hungarian and other Uralic languages.

7.1 Pre-trained models for Hungarian

In this section we present an evaluation of PLMs for Hungarian. We include all models that support Hungarian and were available in the Hugging Face model repository as of September 2020.

Nemeskey (2020a) released the first BERT model for Hungarian named *HuBERT* trained on Webcorpus 2.0 (Nemeskey, 2020a, ch. 4). HuBERT uses the same BERT-base architecture as mBERT and XLM-RoBERTa with a smaller subword vocabulary with 32k subwords. The total parameter count is 110M.

In this section we focus on evaluation for the Hungarian language. We compare HuBERT against multilingual models using three tasks: morphosyntactic probing, POS tagging and NER. We show that HuBERT outperforms all multilingual models, particularly in the lower layers, and often by a large margin. We also show that subword tokens generated by HuBERT's tokenizer are closer to Hungarian morphemes than the ones generated by the other models.

We evaluate the models through three tasks: morphosyntactic probing, POS tagging and NER. Hungarian has a rich inflectional morphology and largely free word order. Morphology plays a key role in parsing Hungarian sentences.

This work was presented at the XVII. Conference on Hungarian Computational Linguistics (MSZNY2021) (Ács et al., 2021). Our data, code and the full result tables are available on GitHub[2].

7.1.1 Evaluation tasks and data

Hungarian is considered a medium resource language in terms of the availability of NLP resources as described in Section 5.4.3.3. However, most of these high level tasks such as question answering were

[1] https://huggingface.co/google-bert/bert-base-german-cased

[2] https://github.com/juditacs/hubert_eval

not yet available at the time of these experiments (late 2020) so we instead focus on three low-level tasks for which the data is readily available or can be automatically generated by high quality taggers.

Morphosyntactic probing Our lowest level task is morphosyntactic probing made possible by high quality morphological analyzers. We use the setup and the dataset introduced in Section 6.2 with some additions. We do not limit the selection of tags to the four tags used in multilingual probing, instead we generate probing data for every possible tag that is frequent enough in Hungarian. This results in 11 tasks listed in Table 7.1. 6 tasks are binary, 3 are 3-way, one is a 4-way and one is an 18-way classification problem. Another change is that our source is not the Hungarian UD Treebank since it is only 42k tokens and therefore not sufficient for our data sampling technique (cf. Section 6.2.2). We use an automatically annotated version of Webcorpus 2.0 for sampling every possible morphological probing task in Hungarian.[3]

Morph tag	**POS**	**#classes**	**Values**
Case	noun	18	Abl, Acc, ..., Ter, Tra
Degree	adj	3	Cmp, Pos, Sup
Mood	verb	4	Cnd, Imp, Ind, Pot
Number[psor]	noun	2	Sing, Plur
Number	adj	2	Sing, Plur
Number	noun	2	Sing, Plur
Number	verb	2	Sing, Plur
Person[psor]	noun	3	1, 2, 3
Person	verb	3	1, 2, 3
Tense	verb	2	Pres, Past
VerbForm	verb	2	Inf, Fin

Table 7.1: List of Hungarian morphological probing tasks.

Sequence tagging tasks Our setup for the two sequence tagging tasks is similar to that of the morphological probes except we train a shared classifier on top of all token representations. Since multiple subwords may correspond to a single token (see Section 7.1.3 for more details), we need to aggregate them in some manner: we pick either the first one or the last one. We also experimented with other pooling methods such as elementwise max and sum but they did not make a significant difference. Our concurrent work in Section 6.4 found some justification for more sophisticated subword pooling methods but due to resource constraints, we did not test all of them. We use two datasets for POS tagging. One is the Szeged Universal Dependencies Treebank (Farkas et al., 2012; Nivre et al., 2018) consisting of 910 train, 441 validation, and 449 test sentences. Our second, larger dataset is a subsample of Webcorpus 2 tagged with emtsv (Indig et al., 2019) with 10,000 train, 2000 validation, and 2000 test sentences. Our architecture for NER is identical to the POS tagging setup. We train it on the Szeged NER corpus consisting of 8172 train, 503 validation, and 900 test sentences.

[3]We use morphosyntactic and morphological probing interchangeably since in the case of Hungarian, morphology is almost solely marked on the word form and syntax plays a very limited role.

7.1.2 The models evaluated

At the time of this study (January 2021), there were 5 models available with Hungarian support in the Hugging Face repository[4]. We compare these 5 models in the following experiments:

HuBERT the Hungarian BERT, is a BERT-base model with 12 Transformer layers, 12 attention heads, each with 768 hidden dimensions and a total of 110 million parameters. It was trained on Webcorpus 2.0 (Nemeskey, 2020a), a 9-billion-token corpus compiled from the Hungarian subset of Common Crawl[5]. Its string identifier in Hugging Face Transformers is `SZTAKI-HLT/hubert-base-cc`.

mBERT the cased version of the multilingual BERT introduced in Section 5.2.2. Its string id is `bert-base-multilingual-cased`.

XLM-RoBERTa the multilingual version of RoBERTa introduced in Section 5.2.3. Its string id is `xlm-roberta-base`.

XLM-MLM-100 is a larger variant of XLM-RoBERTa with 16 instead of 12 layers. Its string id is `xlm-mlm-100-1280`.

distilbert-base-multilingual-cased is a *distilled* version of mBERT. It cuts the parameter budget and inference time by roughly 40% while retaining 97% of the tutor model's NLU capabilities. Its string id is `distilbert-base-multilingual-cased`.

7.1.3 Subword tokenization

Subword tokenization plays an important role in achieving good performance on morphologically rich languages such as Hungarian. Out of the 5 models we compare, HuBERT, mBERT and DistilmBERT use the WordPiece algorithm (Schuster and Nakajima, 2012), XLM-RoBERTa and XLM-MLM-100 use the SentencePiece algorithm (Kudo and Richardson, 2018) introduced in Section 5.3. The multilingual models consist of about 100 languages, and the vocabularies per language are (not linearly) proportional to the amount of training data available per language. Since HuBERT is trained on monolingual data, it can retain less frequent subwords in its vocabulary, while mBERT, RoBERTa and MLM-100, being multilingual models, have token information from many languages, so we anticipate that HuBERT is more faithful to Hungarian morphology. DistilmBERT uses the tokenizer of mBERT, thus it is not included in this section.

Table 7.2 lists statistics about each tokenizer. We use emtsv's segmenter (last column) as an estimate of the *ideal* morphological segmentation. We hypothesize that a good tokenizer should reuse subwords as much as possible particularly in an agglutinative language like Hungarian. We compute the entropy of the distribution of subwords in the first and the last place of a token. The lower the entropy, the more likely is the tokenizer to reuse subwords but this should be evaluated along with the length of the subword in characters. The bottom three rows offer a direct comparison with emtsv's segmenter.

As shown in Table 7.2, there is a gap between the Hungarian and multilingual models in almost every measure. mBERT's shared vocabulary consists only of 120k subwords for all 100 languages while HuBERT's vocabulary contains 32k items and is uniquely for Hungarian. Given the very limited inventory of mBERT, only the most frequent Hungarian words are represented as a single token,

[4] https://huggingface.co/models
[5] https://commoncrawl.org/

	HuBERT	**mBERT**	**XLM-RoBERTa**	**MLM-100**	**emtsv**
Languages supported	1	104	100	100	1
Vocabulary size	32k	120k	250k	200k	–
Entropy of first SW	8.99	6.64	6.33	7.56	8.26
Entropy of last SW	6.82	6.38	5.60	6.89	5.14
More than one SW	94.9%	96.9%	96.5%	97.0%	95.8%
Length in SW or fertility	2.8±1.4	3.9±1.8	3.2±1.4	3.5±1.5	3.1±1.1
Length of first SW	4.3±3.0	2.7±1.9	3.5±2.7	3.1±2.0	5.2±2.4
Length of last SW	3.8±2.9	2.6±1.8	3.1±2.2	2.8±1.8	2.7±1.7
Accuracy to emtsv	0.16	0.05	0.14	0.08	1.00
Accuracy to emtsv in first SW	0.41	0.26	0.44	0.33	1.00
Accuracy to emtsv in last SW	0.43	0.41	0.47	0.39	1.00

Table 7.2: Measures on the train data of the POS tasks. The length of first and last subword (SW) is calculated in characters, while the word length is calculated in subwords.

while longer Hungarian words are segmented, often very poorly. The average number of subwords a word is tokenized into is 2.77 in the case of HuBERT, while all the other models have significantly higher mean length. This does not pose a problem in itself, since the tokenizers work with a given dictionary size and frequent words need not to be segmented into subwords. But in case of words with rarer subwords, the limits of smaller monolingual vocabulary can be observed, as shown in the following example: *szállítójárművekkel* 'with transport vehicles'; *szállító-jármű-vek-kel* 'transport-vehicle-PL-INS' for HuBERT, *sz-ál-lí-tó-já-rm-ű-vek-kel* for mBERT, which found the affixes correctly (since affixes are high in frequency), but have not found the root 'transport vehicle'. Interestingly HuBERT is often outperformed by XLM-RoBERTa which may be due to the very large vocabulary (250k subwords) that XLM-RoBERTa uses.

Distributionally, HuBERT follows the Zipfian distribution more than any other model, as shown in Figure 7.1. Frequency and subword length are in a linear relationship for the HuBERT model, while in case of the other models, the subword lengths does not seem to be correlated with the log frequency rank. The area of the violins also show that words typically consist of more than 3 subwords for the multilingual models, contrary to the HuBERT, which segments the words typically into one or two subwords.

7.1.4 Results

We find that HuBERT outperforms all models in all tasks, often by a large margin, particularly in the lower layers. As for the choice of subword pooling (first or last) and the choice of layer, we note some trends in the rest of this section.

Morphology The last subword is always better than the first subword except for a few cases for degree ADJ. This is not surprising because superlative is marked with a circumfix and it is differentiated from comparative by a prefix. The rest of the results in this subsection all use the last subword.

HuBERT is better than all models, especially in the lower layers in morphological tasks, as shown in Figure 7.2. However, this tendency starts at the second layer and the first layer does not usually outperform the other models. In some morphological tasks HuBERT systematically outperforms

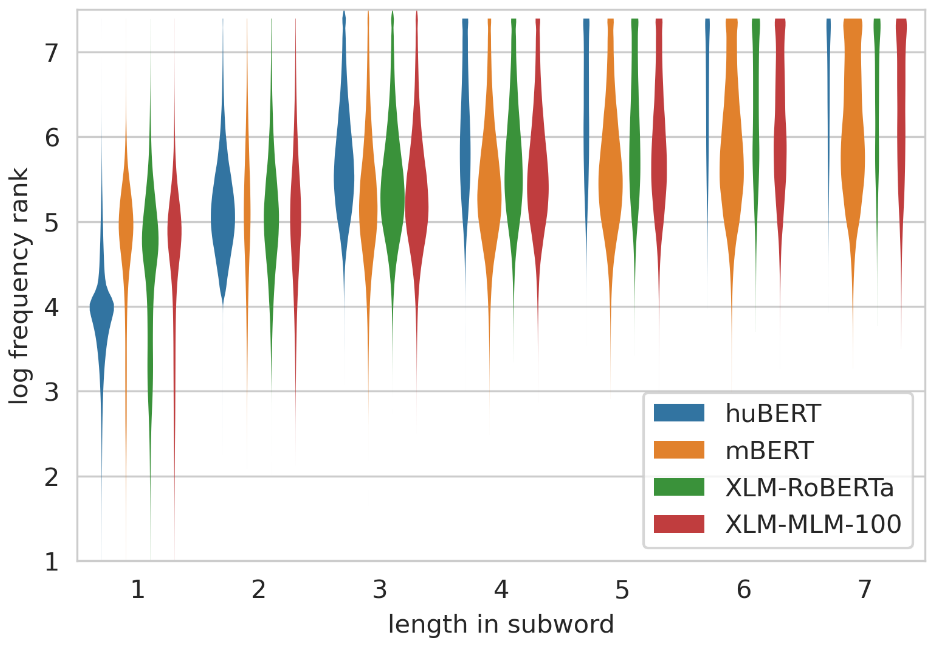

Figure 7.1: Distribution of length in subword vs. log frequency rank. The count of words for one subword length is proportional to the size of the respective violin.

the other models: these are mostly the simpler noun and adjective-based probes. In possessor tasks (tagged `[psor]` in Figure 7.2) XLM-RoBERTa models are comparable to HuBERT, while mBERT and DistilmBERT generally perform worse then HuBERT. In verb tasks XLM-RoBERTa achieves similar accuracy to HuBERT in the higher layers, while in the lower layers, HuBERT tends to have a higher accuracy.

HuBERT is also better than all models in almost all tasks when we use the weighted average of all layers as illustrated by Figure 7.3. The only exceptions are adjective degrees and the possessor tasks. A possible explanation for the surprising effectiveness of XLM-MLM-100 is its higher layer count.

POS tagging Figure 7.4 shows the accuracy of different models on the gold-standard Szeged UD and on the silver-standard data created with emtsv. Last subword pooling always performs better than first subword pooling. As in the morphology tasks, the XLM models perform only a bit worse than HuBERT. mBERT is very close in performance to HuBERT, unlike in the morphological tasks, while DistilmBERT performs the worst, possibly due to its far lower parameter count.

We next examine the behavior of the layers by relative position.[6] The embedding layer is a static mapping of subwords to an embedding space with a simple positional encoding added. Contextual information is not available until the first layer. The highest layer is generally used as the input for downstream tasks. We also plot the performance of the middle layer. As Figure 7.5 shows, the embedding layer is the worst for each model and, somewhat surprisingly, adding one contextual layer only leads to a small improvement. The middle layer is actually better than the highest layer which confirms the findings of Tenney et al. (2019a) that BERT rediscovers the NLP pipeline along its layers, where POS tagging is a mid-level task. As for the choice of subword, the last one is generally better, but the gap shrinks as we go higher in layers.

[6]We only do this on the smaller Szeged dataset due to resource limitations.

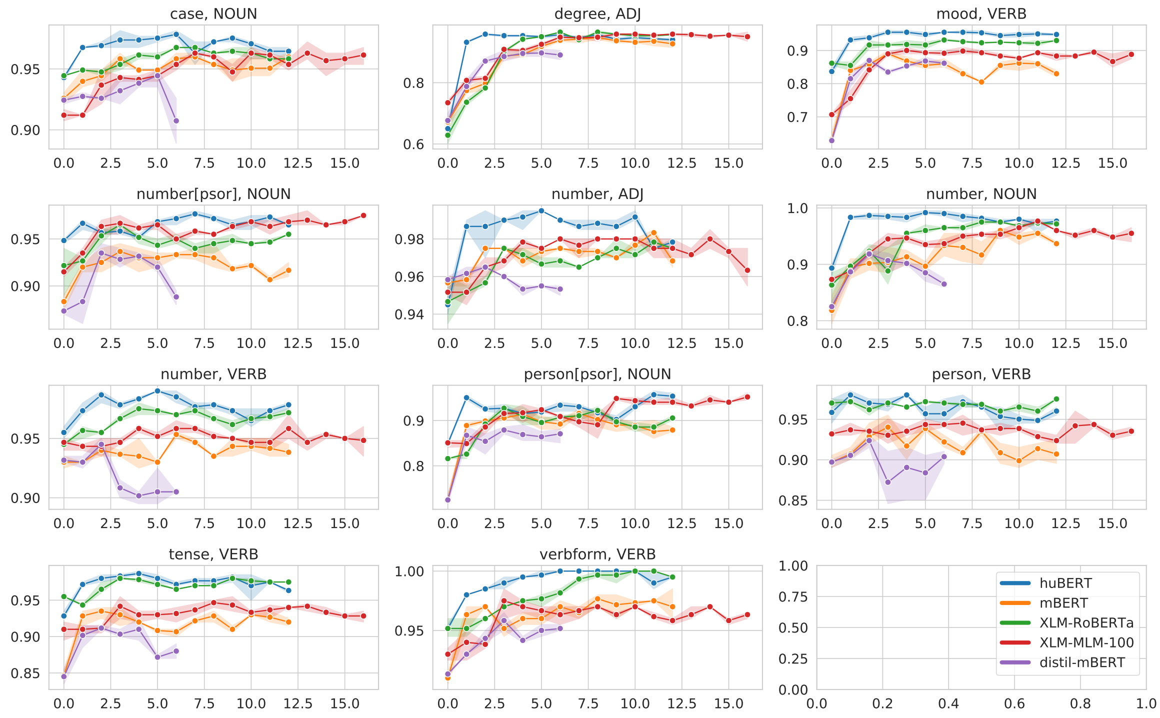

Figure 7.2: The layerwise accuracy of morphological probes using the last subword. Shaded areas represent confidence intervals over 3 runs.

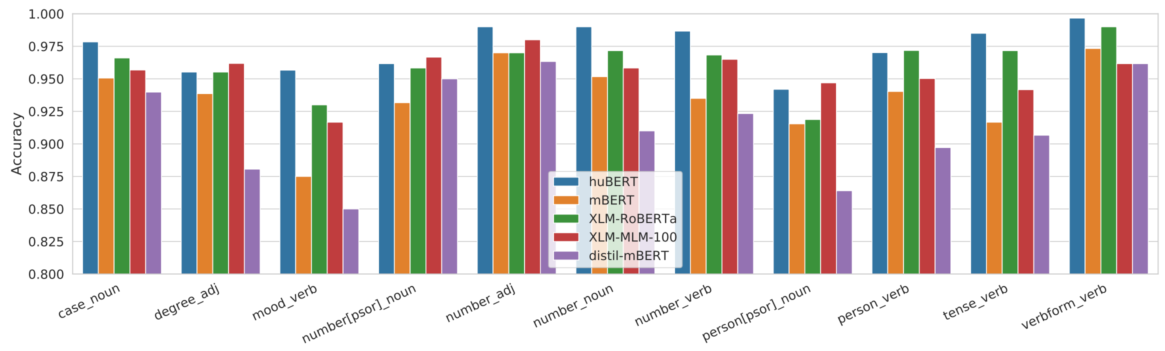

Figure 7.3: Probing accuracy using the weighted sum of all layers.

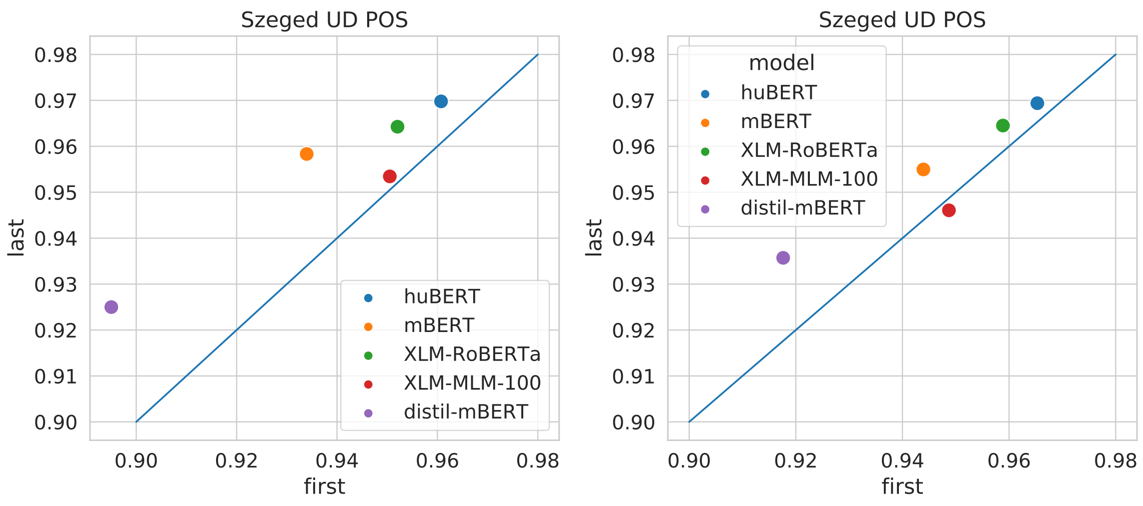

Figure 7.4: POS tag accuracy on Szeged UD and on the Webcorpus 2.0 sample

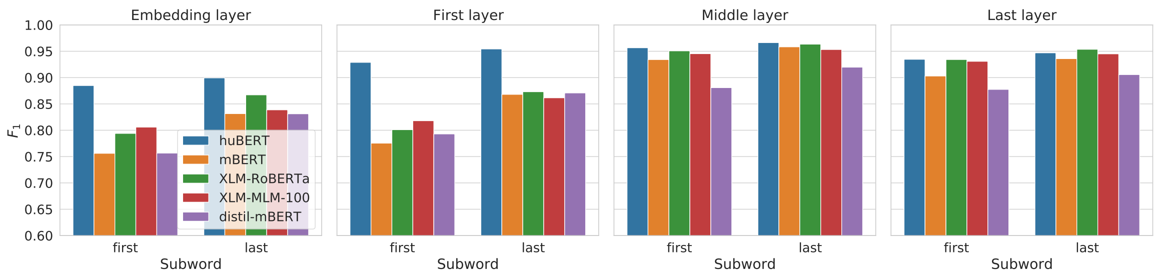

Figure 7.5: Szeged POS at 4 layers: embedding layer, first Transformer layer, middle layer, and highest layer.

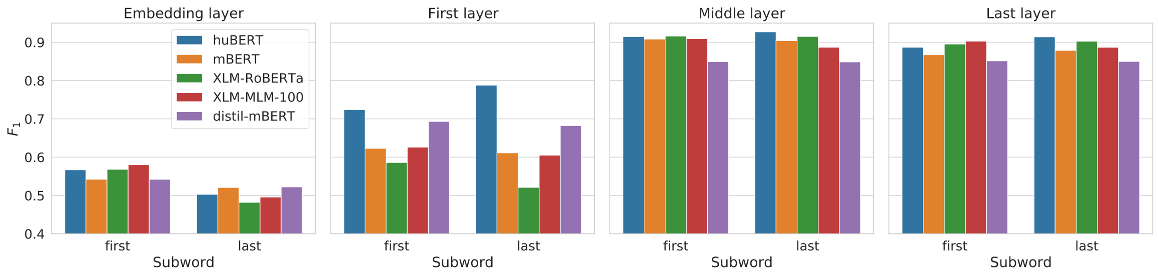

Figure 7.6: NER F_1 score at the lowest, middle and highest layers.

Named entity recognition In the NER task (Figure 7.6), all of the models perform very similarly in the higher layers, except for DistilmBERT which has nearly 3 times the error of the best model, HuBERT. The closer we get to the global optimum, the clearer HuBERT's superiority becomes. Far away from the optimum, when we use only the embedding layer, first subword is better than last, but the closer we get to the optimum (middle and last layer), the clearer the superiority of the last subword choice becomes.

7.1.5 Conclusion

In this section we showed that the current generation of PLMs have good support for Hungarian on low and mid-level tasks. HuBERT, a BERT model specific to Hungarian, is the best contender, especially at the lower layers, but XLM-RoBERTa is a close competitor and mBERT also performs reasonably well. We also demonstrated that the subword tokenizer of HuBERT and to some extent that of XLM-RoBERTa segment somewhat similarly to a high quality rule based morphological segmenter. Lastly, we found that using the last subword for token-level tasks is always the best choice for Hungarian, except for cases where discontinuous morphology is involved, as in circumfixes and infixed plural possessives (Antal, 1963; Mel'cuk, 1972).

7.2 Pre-trained models for Uralic languages

In this section we extend the evaluation method used for Hungarian (Section 7.1) to other members of the Uralic languages. This work was presented at the Seventh International Workshop on Computational Linguistics of Uralic Languages (IWCLUL2021) (Ács et al., 2021b). The code, the data and the full result tables are available on GitHub[7].

Here we evaluate monolingual, multilingual, and randomly initialized language models from the BERT family on a variety of Uralic languages including Estonian, Finnish, Hungarian, Erzya, Moksha, Karelian, Livvi, Komi Permyak, Komi Zyrian, Northern Sámi, and Skolt Sámi.

Uralic languages have received relatively moderate interest from the language modeling community. Aside from the three national languages, Estonian, Finnish and Hungarian, no other Uralic language is supported by any of the multilingual models, nor does any have a monolingual model. There are no Uralic languages among the 15 languages of XNLI. Wu and Dredze (2020) do explore all 100 languages that mBERT supports but do not go into monolingual details.

In this section we evaluate multilingual and monolingual models on Uralic languages. We consider the same three evaluation tasks as in Section 7.1: morphosyntactic probing, POS tagging and NER. We also use the models in a crosslingual setting, in other words, we test how monolingual models perform on related languages. We show that

- these language models are very good at all three tasks when fine-tuned on a small amount of task specific data,
- for morphological tasks, when native BERT models are available (et, fi, hu), these outperform the others on their native language, though the advantage over XLM-RoBERTa is not statistically significant,
- for POS and NER, the use of native models from related, even closely related languages, rarely brings improvement over the multilingual models or even English models,

[7] https://github.com/juditacs/uralic_eval

- as long as the alphabet that the language uses is covered in the vocabulary of the model, we can transfer mBERT (or RuBERT) to the NER and POS tasks with surprisingly little fine-tuning data,
- our few-shot models for minority Uralic languages appear to be state of the art POS and NER models without special efforts toward hyperparameter optimization if there is sufficient fine-tuning data.

7.2.1 Data

Morphosyntactic probing We generate the probing data for Estonian and Finnish from the Universal Dependencies (UD) Treebanks and from the automatically tagged Webcorpus 2.0 for Hungarian since the Hungarian UD is very small. Unfortunately we could not extend the list of languages to other Uralic languages because their treebanks are too small to sample enough data with the sampling method we introduced in 6.2.2. We were able to generate enough probing data for 11 Estonian, 16 Finnish and 11 Hungarian tasks, see Table 7.6 for the full list of these.

Sequence tagging tasks Our setup for the two sequence tagging tasks is similar to that of the morphological probes except we train a shared classifier on top of all token representations. We use the vector corresponding to the first subword in both tasks. Although this may be suboptimal in morphology, Ács et al. (2021a) showed that the difference is smaller for POS and NER. We also fine-tune the models which seems to close the gap between first and last subword pooling for morphology. For sequence tagging tasks, unlike for morphology, we found that the weighted average of all layers is suboptimal compared to simply using the top layer, so the experiments presented here all use the top layer.

POS tagging We sample 2000 train, 200 validation and 200 test sentences as POS training data from the largest UD treebank in Estonian and Finnish, and from Webcorpus 2.0 for Hungarian. Aside from these three, Erzya, Moksha, Karelian, Livvi, Komi Permyak, Komi Zyrian, Northern Sami, and Skolt Sami have UD treebanks, but these are considerably smaller in size. Although none of these languages are officially supported by any of the language models we evaluate, we train crosslingual models and find that the models have remarkable crosslingual capabilities.

NER Our NER data is sampled from WikiAnn (Pan et al., 2017). WikiAnn has data in Erzya, Estonian, Finnish, Hungarian, Komi Permyak, Komi Zyrian, Moksha, and Northern Sami.[8] Similarly to the POS training data, we sample 2000 training, 200 validation and 200 test sentences when available, see Table 7.3 for actual training set sizes.

7.2.2 The models evaluated

Our goal is twofold: we want to assess monolingual models against multilingual models, and we want to evaluate the models on 'unsupported' languages, both typologically related and unrelated.

We pick two multilingual models, mBERT and XLM-RoBERTa. Our choices for monolingual models are EstBERT for Estonian, FinBERT for Finnish and HuBERT for Hungarian. As a control, we also

[8]WikiAnn also has Udmurt data, but the transcription is problematic: Latin and Cyrillic are used inconsistently, Wikipedia Markup is parsed incorrectly etc.

Language	Code	Script	Morph	POS	NER
Hungarian	[hu]	Latin	26k	2000	2000
Finnish	[fi]	Latin	38k	2000	2000
Estonian	[et]	Latin	26k	2000	2000
Erzya	[myv]	Cyrillic	0	1680	1800
Moksha	[mdf]	Cyrillic	0	164	400
Karelian	[krl]	Latin	0	224	0
Livvi	[olo]	Latin	0	122	0
Komi Permyak	[koi]	Cyrillic	0	78	2000
Komi Zyrian	[kpv]	Cyrillic	0	562	1700
Northern Sami	[sme]	Latin	0	2000	1200
Skolt Sami	[sms]	Latin	0	101	0

Table 7.3: Size of training data for each language.

test the English BERT as a general test for cross-language transfer. Since many Uralic speaking communities are in Russia and the languages are heavily influenced by Russian, we test RuBERT on these languages. Finally, we also test a randomly initialized mBERT. We do this because the capacity of the BERT-base models is so large that they may memorize the probing data alone (cf. Section 6.6.4). Many models have cased and uncased version, the latter often removing diacritics along with lowercasing. Since diacritics play an important role in many Uralic languages, we mainly use the cased models. We return to this issue in 7.2.3. See Table 7.4 for more details on the models.

The evaluated models, all from the BERT/RoBERTa family, differ only in the choice of training data and the training objective. Their subword vocabularies are also derived from the training data. They all have 12 Transformer layers, with 12 heads, and 768 hidden dimensions, for a total of 108M–177M parameters with the exception of XLM-RoBERTa which has a larger vocabulary and therefore a much higher parameter count (278M). The models along with their string identifier are summarized in Table 7.4.

Model	Identifier	Language(s)	Reference
mBERT	bert-base-multilingual-cased	100+ inc. et, fi, hu	Devlin et al. (2019)
XLM-RoBERTa	xlm-roberta-base	100 inc. et, fi, hu	Liu et al. (2019b)
EstBERT	tartuNLP/EstBERT	Estonian	Tanvir et al. (2021)
FinBERT	TurkuNLP/bert-base-finnish-cased-v1	Finnish	Virtanen et al. (2019)
HuBERT	SZTAKI-HLT/hubert-base-cc	Hungarian	Nemeskey (2020a)
EngBERT	bert-base-cased	English	Devlin et al. (2019)
RuBERT	DeepPavlov/rubert-base-cased	Russian	Kuratov and Arkhipov (2019)
rand-mBERT	mBERT with random weights	any	described in Section 7.2.2

Table 7.4: List of models we evaluate.

7.2.3 Subword tokenization

The native models, trained on monolingual data, have longer and more meaningful subwords (see the bolded entries in Table 7.5). This greatly facilitates the sharing of train data, a matter of great importance for Uralic languages where there is little text available to begin with.

	mBERT	**XLM-RoBERTa**	**EstBERT**	**FinBERT**	**HuBERT**	**RuBERT**	**EngBERT**
Vocabulary size	120k	250k	50k	50k	32k	120k	29k
Missing [et] (%)	.0	.0	**.2**	.0	.5	.1	.2
Missing [fi] (%)	.0	.0	.0	**.0**	.4	.0	.0
Missing [hu] (%)	.1	.0	21.5	48.3	**.1**	2.7	.2
Missing [sme] (%)	.2	.0	15.0	47.4	5.1	4.8	.2
Missing [myv] (%)	.0	.0	97.5	97.5	97.5	.0	.0
Subword length [et]	3.7±1.4	4.2±1.7	**5.8**±2.6	3.7±1.4	3.1±1.2	3.1±1.2	3.5±1.4
Subword length [fi]	3.8±1.4	4.5±1.9	3.8±1.4	**5.9**±2.5	3.1±1.1	3.1±1.1	3.4±1.4
Subword length [hu]	3.5±1.5	4.2±2.0	3.3±1.2	3.1±1.1	**5.0**±2.4	3.0±1.1	3.3±1.4
Subword length [sme]	3.2±1.0	3.4±1.1	3.2±1.1	3.2±1.1	3.1±1.2	2.9±1.0	3.0±1.0
Subword length [myv]	3.1±1.2	3.2±1.0	1.0±0.0	1.0±0.0	1.0±0.0	3.4±1.2	1.1±0.4
Character length [et]	9.2	9.2	**9.2**	9.2	9.2	9.2	9.2
Character length [fi]	9.3	9.3	9.3	**9.3**	9.3	9.3	9.3
Character length [hu]	9.8	9.8	9.6	8.8	**9.8**	9.8	9.9
Character length [sme]	8.5	8.5	8.3	7.6	8.5	8.4	8.5
Character length [myv]	7.3	7.3	1.8	1.8	1.7	7.3	7.3
Fertility [et]	3.4	2.8	**2.1**	3.6	4.4	4.3	4.3
Fertility [fi]	3.3	2.7	3.5	**1.9**	4.6	4.4	4.5
Fertility [hu]	4.0	3.2	5.2	4.5	**2.8**	5.4	5.6
Fertility [sme]	3.7	3.6	4.1	3.3	4.5	4.6	4.7
Fertility [myv]	3.6	3.3	1.1	1.1	1.1	3.0	7.2

Table 7.5: Major characteristics of cross-language tokenization. Boldface font marks the corresponding language-model pairs. For a language-language code correspondence, see Table 7.3.

Both BERT- and RoBERTa-based models first tokenize along whitespaces, but the handling of missing characters differs significantly. In BERT-based models, if there is a character missing from the tokenizer's vocabulary, the model discards the whole segment between whitespaces, labeling it [UNK]. In cross-lingual cases many words are lost since monolingual models' vocabularies tend to lack the extra characters of a different language. In contrast, XLM-RoBERTa deletes the unknown characters, but the string that remains between whitespaces is segmented, so the loss of information is not as severe.

Table 7.5 summarizes different measures in language-model pairs. As a general observation, Latin script models (FinBERT, HuBERT, EstBERT) are unusable on Cyrillic text, as seen e.g. on Erzya, where Latin script models produce [UNK] token for 97.5% of the word types. This is also seen for Northern Sami and Hungarian, which have many non-ascii characters, see the Hungarian-EstBert/FinBERT pairs and the Northern Sami-FinBERT/HuBERT pairs.

The mean subword length generally lies between 3.0 and 3.5 for most pairs - naturally, the corresponding language-model pairs have much higher mean subword length, 5.0 to even 5.9. This range is true not only for Latin script languages, but for Cyrillic script languages as well, as indicated by Erzya, which has a mean subword length of 3.1 to 3.4 on the multilingual models and on RuBERT.

Fertility (Ács, 2019) is defined as the average number of BERT word pieces found in a single real word type. EstBERT on Estonian and FinBERT on Finnish have very similar fertility values (2.1 and 1.9), but HuBERT on Hungarian has much higher fertility. This is mainly caused by the different vocabulary sizes - the Finnic models have 50000 subwords in their vocabulary, HuBERT only contains 32000 subwords. The rest of the fertility values are mostly over 3. In extreme cases, a word is segmented into letters, which is the case for EngBERT on Erzya, but the non-Hungarian models on

Hungarian also produce very high fertility values.

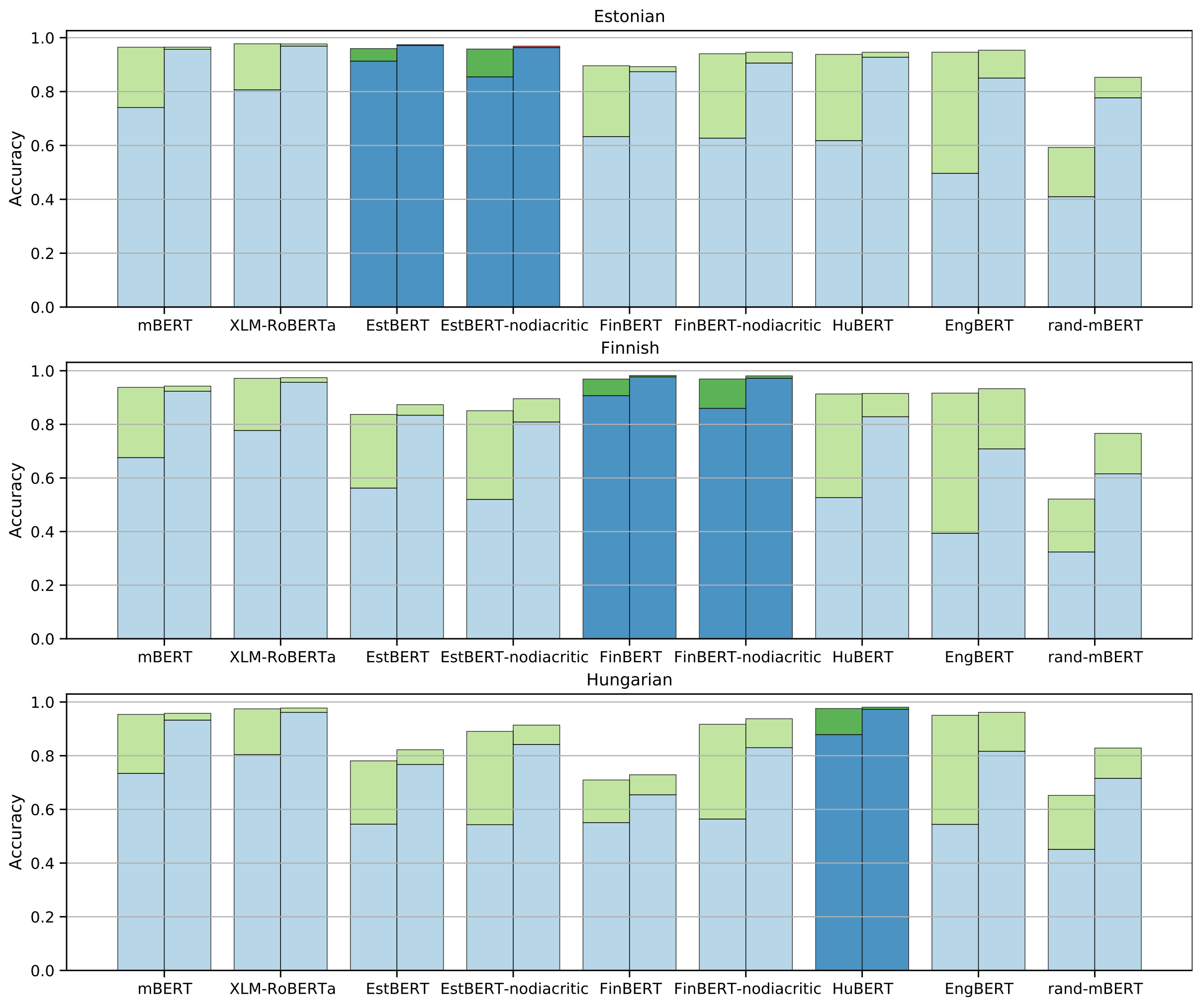

Figure 7.7: Mean accuracy of morphological tasks by language. The bars are grouped in two, the left one is the result of probing the first subword, the right one is the results of probing the last subword. Blue bars are without fine-tuning, green bars are with fine-tuning. Monolingual models are highlighted.

7.2.4 Results

7.2.4.1 Morphology

Morphological tasks are generally easy for most models and we see reasonable accuracy from crosslingual models as illustrated by Figure 7.7. Mean accuracies, especially after fine-tuning, are generally above 90%, except, unsurprisingly, for the randomly initialized models.

Subword choice We first start by examining the choice of subword on morphological tasks. We try probing the first and the last subword and we find that there is a substantial gap in favor of the last subword. This is unsurprising considering that Uralic languages are mainly suffixing. This gap on average shrinks from 0.21 to 0.032 when we fine-tune the models on the probing data (Figure 7.7

shows this gap in green). Similarly to our findings in Section 7.1.4, without fine-tuning there is only one task, ⟨Hungarian, Degree, ADJ⟩, where probing the first subword is better than probing the last one for some models. This is explained by the fact that the superlative in Hungarian is formed from the comparative by a prefix.

Monolingual models are only slightly better than the two multilingual models, XLM-RoBERTa in particular. We run paired t-tests on the accuracy of each model pair over the 11 (et, hu) or 16 (fi) morphological tasks in a particular language and find that the difference between the monolingual model and XLM-RoBERTa is never significant, and for Estonian, neither is the difference between EstBERT and mBERT.

Cross-lingual transfer works only if we fine-tune the models. Interestingly, language relatedness does not seem to play a role here. FinBERT transfers worse to Estonian than HuBERT, and EstBERT transfers worse to Finnish than HuBERT. Interestingly, EngBERT transfers better to all three models than the other native BERTs, and for Finnish and Hungarian it is actually on par with mBERT.

Diacritics. As seen from the first panel of Table 7.5, EstBERT and FinBERT replace words with unknown characters with [UNK] to such an extent that a large proportion of types end up being filtered. We try to mitigate this issue by preemptively removing all diacritics from the input (EstBERT-nodiacritic and FinBERT-nodiacritic). It appears that this has little effect on the original language, but cross-lingual transfer is improved for Finnish. In the sequence tagging tasks that we now turn to, we remove the diacritics when we evaluate EstBERT or FinBERT in a cross-lingual setting.

7.2.4.2 POS and NER

We extend our studies to all Uralic languages with any training data (see Table 7.3) and we limit the discussion to fine-tuned models since cross-lingual transfer does not work without fine-tuning. We split the languages into two groups, Latin and Cyrillic, and we only test models with explicit support for the script that the language uses. Multilingual models support both scripts. Figures 7.8 and 7.9 show the results by language.

National languages. We generally find the best performance in the three languages with native support: Estonian, Finnish and Hungarian. Monolingual models perform the best in their respective language but the two multilingual models are also very capable.

Cross-lingual transfer does not seem to benefit from language relatedness, EngBERT transfers just as well as other monolingual models. Even extremely close relatives such as Livvi and Finnish do not transfer better than XLM-RoBERTa to Livvi. On the other hand, FinBERT is the best for Karelian POS, another close relative of Finnish. The writing system and shared vocabulary also seem to play an important role, as seen from RuBERT's usefulness on unrelated but Cyrillic-using Uralic languages, illustrated in Figure 7.9.

XLM-RoBERTa is generally a strong model for cross-lingual transfer for all Uralic languages. We suspect that this is due to its large subword vocabulary, which may provide a better generalization basis for capturing the orthographic cues that are often highly indicative in agglutinative languages.

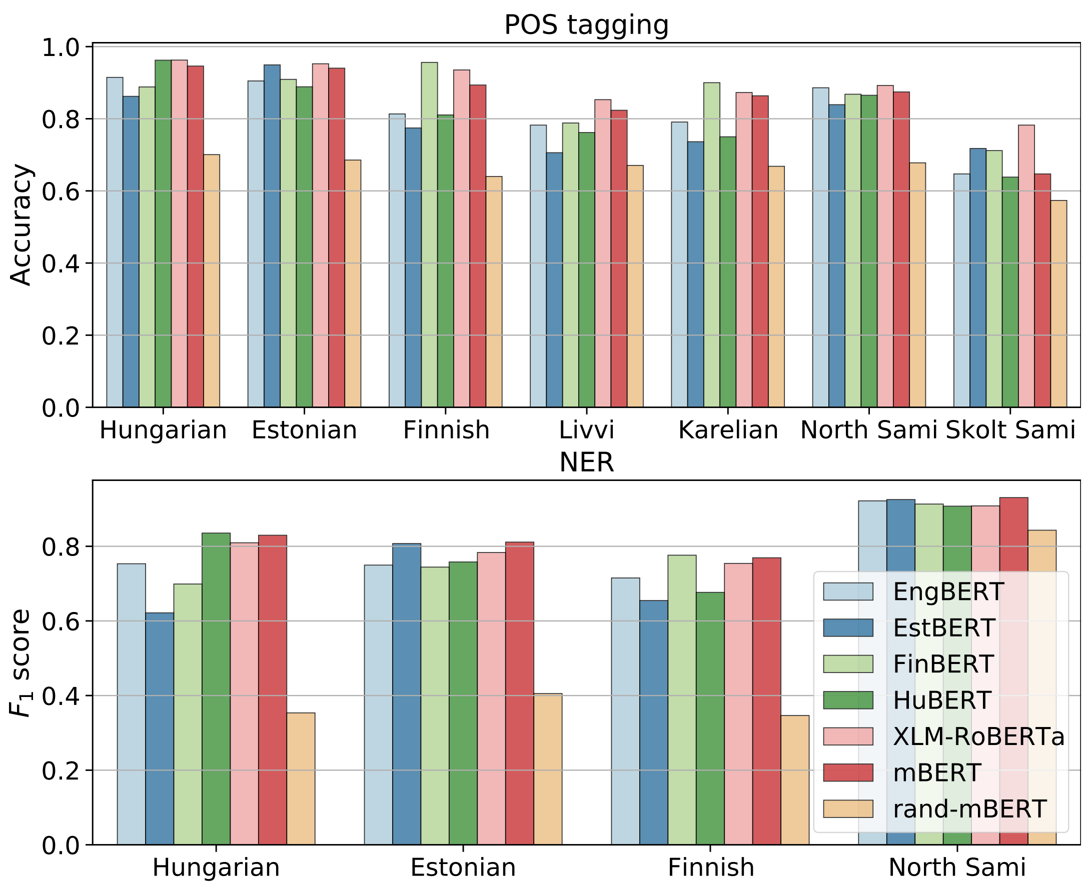

Figure 7.8: POS and NER results on languages that use the Latin alphabet.

North Sami Both POS and NER in North Sami are relatively easy as long as the orthographic cues can be captured (i.e. the Latin script is supported). rand-mBERT is surprisingly successful at NER in North Sami, suggesting that orthographic cues (rand-mBERT uses mBERT's tokenizer) are highly predictive of named entities in North Sami.

7.2.5 Conclusion

Altogether we find that it is possible, and relatively easy, to transfer models to new languages with fine-tuning on very limited training data, though extremely limited data still hinders progress: compare Erzya (1680 train sentences) to Moksha (164 train sentences) in Figure 7.9.

EngBERT and RuBERT, which we introduced as a control for language transfer among genetically unrelated languages, transfer quite well: in particular the Latin-script EngBERT transfers better to Hungarian than FinBERT or EstBERT.

We note that we did not perform monolingual hyperparameter search or any preprocessing, and there is probably room for improvement for each of these languages. The biggest immediate gains are expected from extending the UD and WikiAnn datasets, and from careful handling of low-level characterset and subword tokenization issues. There are many Uralic languages that still lack basic resources, in particular the entire Samoyedic branch, Mari, and Ob-Ugric languages, are currently out of scope. Another avenue of research could be to work towards a stronger mBERT Interlingua, or perhaps one for each script family, as the character set issues are clearly relevant.

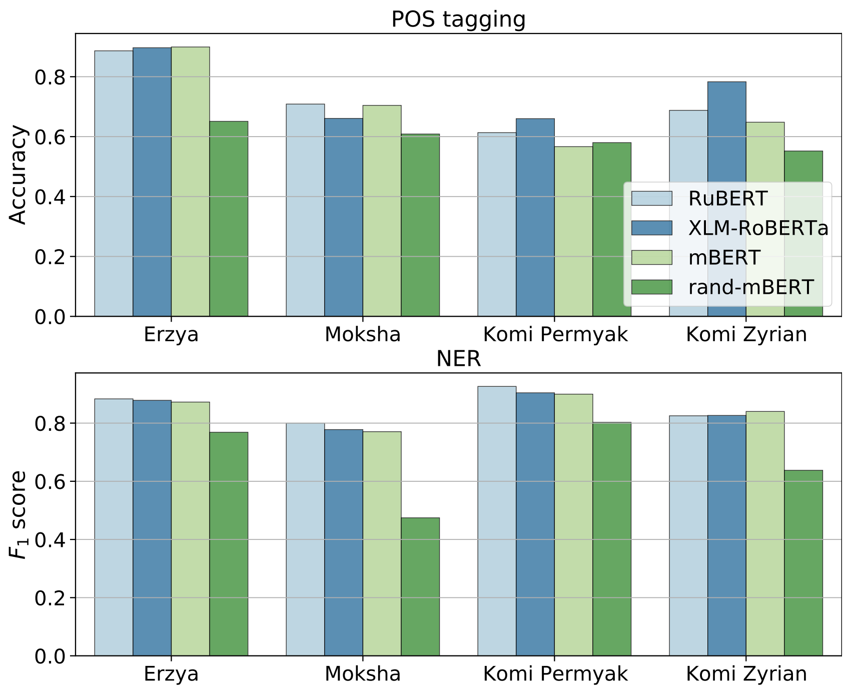

Figure 7.9: POS and NER results on languages that use the Cyrillic alphabet.

Morph tag	**POS**	**Estonian**	**Finnish**	**Hungarian**
Case	adj	8 classes	11 classes	
Case	noun	15 classes	12 classes	18 classes
Case	propn		8 classes	
Case	verb		12 classes	
Degree	adj		Cmp, Pos, Sup	Cmp, Pos, Sup
Derivation	adj		Inen, Lainen, Llinen, Ton	
Derivation	noun		Ja, Lainen, Minen, U, Vs	
InfForm	verb		1, 2, 3	
Mood	verb		Cnd, Imp, Ind	Cnd, Imp, Ind, Pot
Number psor	noun			Sing, Plur
Number	a/n/v	Sing, Plur	Sing, Plur	Sing, Plur
PartForm	verb		Pres, Past, Agt	
Person psor	noun			1, 2, 3
Person	verb	1, 2, 3		1, 2, 3
Tense	adj	Pres, Past		
Tense	verb	Pres, Past	Pres, Past	Pres, Past
VerbForm	verb	Conv, Fin, Inf, Part, Sup	Inf, Fin, Part	Inf, Fin
Voice	adj	Act, Pass		
Voice	verb	Act, Pass	Act, Pass	

Table 7.6: List of morphological probing tasks.

Chapter 8

Perturbations and Shapley values

In the previous chapters we showed that PLMs exhibit proficiency in morphosyntax and most morphosyntactic tasks do not pose a serious challenge for PLMs. In this chapter we aim to find where morphosyntactic information is available in the sentence. We do this by systematically removing some information from the probing sentence and retraining the probe on the modified sentences. The drop in performance is indicative of the importance of the information we removed.

We then further elaborate by assigning importance to the individual tokens of the sentence relative to the target token. We treat the individual tokens as players of a game and compute the Shapley values, an additive measure of importance of each player.

Although we perform the experiments required for computing the Shapley values for both mBERT and XLM-RoBERTa as well as chLSTM, it turns out that most of our observations are similar for mBERT and XLM-RoBERTa, and often for chLSTM, a non-pretrained character-level baseline. This resemblance prompts us to believe that these are somehow characteristics of languages rather than language models. Indeed we often find linguistic explanations for the most surprising results.

Throughout this chapter all methods we present require masking certain tokens, therefore we opt to use the term *masked language model* rather than the broader term *pre-trained language model*. We examine two multilingual models, mBERT and XLM-RoBERTa along with a non-pretrained baseline, chLSTM, which we extend with an artificial mask token.

8.1 Perturbations

In Section 8.1.1 we analyze the MLMs' knowledge of morphology in more detail through a set of *perturbations* that remove some source of information from the probing sentence. We compare the different perturbations to the unperturbed mBERT, but observe that perturbations often reduces performance to the level of the contextual baseline (chLSTM) or even below. The effect of major perturbations is unmistakable. Table 8.1 exemplifies each perturbation.

Target masking Languages with rich inflectional morphology tend to encode most, if not all, morphological information in the word form alone. We test this by hiding the word form, while keeping the rest of the sentence intact. Recall that BERT is trained with a cloze-style language modeling objective, i.e., 15% tokens are replaced with a `[MASK]` token and the goal is to predict these. We employ

Method	Explanation	Example
Original		Then he ripped open Hermione 's letter and **read** it out loud .
TARG	mask target word	Then he ripped open Hermione 's letter and **[M]** it out loud .
L_2	mask previous 2 words	Then he ripped open Hermione 's [M] [M] **read** it out loud .
R_2	mask next 2 words	Then he ripped open Hermione 's letter and **read** [M] [M] loud .
B_2	mask 2 on each side	Then he ripped open Hermione 's [M] [M] **read** [M] [M] loud .
PERMUTE	shuffle word order	and open **read** Then letter . it out he ripped 's Hermione loud

Table 8.1: List of perturbation methods with examples. The target word is in **bold**. The mask symbol is abbreviated as [M].

this mask token to hide the target word (TARG) from the auxiliary classifier. This means that all orthographic cues present in the word form are removed.[1]

Context masking Many languages encode morphology in short phrases that span a few words, e.g., person/number agreement features on a verb that is immediately preceded by a subject. The verb tense of *read*, while ambiguous on its own, can often be disambiguated by looking at a few surrounding words, such as the presence of an auxiliary (*didn't*), or a temporal expression. We use the relative position of a token to the target word, left context refers to the part of the sentence before the target word, while right context refers to the part after it. We try masking the left (L_N), the right (R_N), and both sides (B_N), where N refers to the number of masked tokens. We expand this analysis using Shapley values in Section 8.2.

Permute Many languages have strict constraints on the order of words. A prime example is English, where little morphology is present at the word level, but reordering the words can change the meaning of a sentence dramatically. Consider the examples *Mary loves John* versus *John loves Mary*: in languages with case inflection the distinction is made by the cases rather than the word order. It has been shown (Sinha et al., 2021; Ettinger, 2020) that BERT models are sensitive to word order in a variety of English and Mandarin tasks. We quantify the importance of word order by shuffling the words in the sentence.

8.1.1 Results

Perturbations change the input sequence in a way that removes information and should result in a decrease in probing accuracy. Since the net changes caused by masking are often quite small[2], particularly for verbs, we define the *effect* of perturbation p on task t when probing model m as:

$$E(m, t, p) = 1 - \frac{\text{Acc}(m, t, p)}{\text{Acc}(m, t)}, \tag{8.1}$$

where $\text{Acc}(m, t)$ is the unperturbed probing accuracy on task t by model m. We present the effect values as percentages of the original accuracy. 50% effect means that the probing accuracy is reduced by half. Negative effect means that the probing accuracy *improves* due to a perturbation.

[1]We use a single mask token regardless of how many wordpieces the target word would contain, rather than masking each wordpiece. Our early experiments showed negligible difference between the two choices.

[2]We run every experiment 10 times and the standard deviation of identical runs is small.

Perturbation	mBERT accuracy	XLM-R accuracy	chLSTM accuracy	mBERT effect	XLM-R effect	chLSTM effect
unperturbed	90.40	91.80	85.00	0.00%	0.00%	0.00%
TARG	75.80	80.00	60.20	16.18%	12.74%	28.93%
PERMUTE	84.60	86.10	77.40	6.74%	6.42%	9.68%
L_2	88.40	90.30	84.20	2.28%	1.63%	1.08%
R_2	89.10	90.90	85.10	1.48%	1.01%	−0.17%
B_2	86.00	87.90	83.80	5.03%	4.42%	1.55%

Table 8.2: Perturbation results by model averaged over 247 tasks. Effect is defined in Equation 8.1.

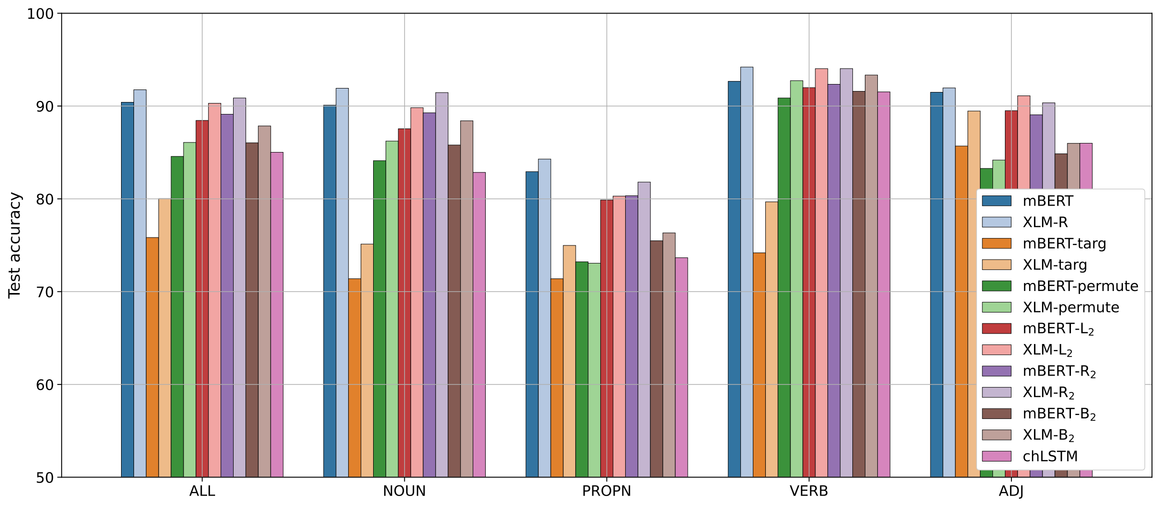

Figure 8.1: Test accuracy of the perturbed probes grouped by POS. The first group is the average of all 247 tasks. The first two bars in each group are the unperturbed probes' accuracy.

Given the large number of tasks and multiple perturbations, instead of listing all individual data points, we average the results over POS, tags, and language families, and point out the main trends and outliers. The overall average perturbation results are listed in Table 8.2.

Our main group of perturbations involves masking one or more words in the input sentence. Both models have dedicated mask symbols, which we use to replace certain input words. In particular, TARG masks the target word, where most of the information is contained – precisely how much will be discussed in Section 8.2. PERMUTE shuffles the entire context, leaving the target word fixed, L_2 masks the two words preceding that target word, R_2 masks the two words following the target and B_2 masks both the preceding two and the following two words. Remarkably, PERMUTE and B_2 are highly correlated, a matter we shall return to in 8.1.1.2. Figure 8.1 shows the average test accuracy of the probes by perturbation grouped by POS.

8.1.1.1 Context masking

Proper nouns seem to be affected the most by context masking perturbations. This is probably caused by the lack of morphological information in the word form itself, at least in Slavic languages, where

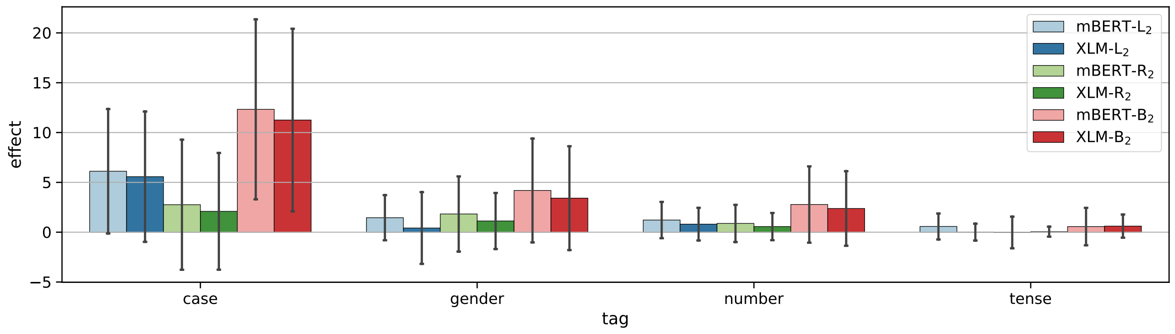

Figure 8.2: The effect of context masking perturbations by tag. Error bars indicate the standard deviation.

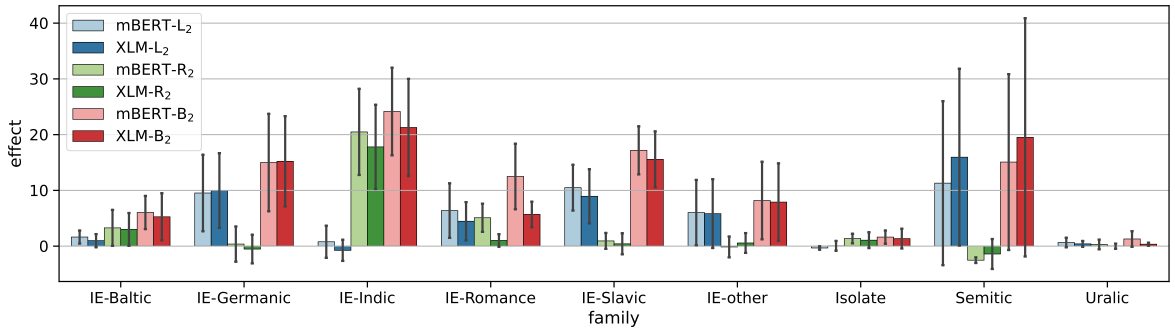

Figure 8.3: The effect of context masking on case tasks by language family. Error bars indicate the standard deviation.

proper nouns are often indeclinable[3]. The models pick up much of the information from the context. We shall examine this in more detail in Section 8.2.

Although the average effect is rather modest, there are some tasks that are affected significantly by context masking perturbations. Figure 8.2 shows the effect (as defined in Equation 8.1) by tag.

Since case is affected the most, we examine it a little closer. Figure 8.3 shows the effect of context masking on case tasks grouped by language family. Uralic results are barely affected by context masking, which confirms that the target word alone is indicative of the case in Uralic languages. Germanic, Semitic, and Slavic case probes are moderately affected by L_2 and somewhat surprisingly, we find a small improvement in probing accuracy, by R_2. Indic probes are the opposite, R_2 has over 20% effect, while L_2 is close to 0. Indic word order is quite complex, with a basic SOV word order affected both by split ergativity and communicative dynamism (topic/focus) effects (Jawaid and Zeman, 2011). Again we suspect that these complexities overwhelm the MLMs, which work best with mountains of data, typically multi-gigaword corpora, 3-4 orders of magnitude more than what can reasonably be expected from primary linguistic data, less than thirty million words during language acquisition (Hart and Risley, 1995).

[3]Slavic tasks account for roughly one third of the probing tasks, see Figure 6.2.

8.1.1.2 Target masking and word order

We discuss TARG and PERMUTE in conjunction since they often have inverse effect for certain languages and language families. Target masking or TARG is by far the most destructive perturbation with an average effect of 16.1% for mBERT and 12.7% for XLM-RoBERTa. PERMUTE is also a significant perturbation, particularly for case tasks and adjectives. As Figure 8.4 shows, the effects differ widely among tasks but some trends are clearly visible. TARG clearly plays an important role in many if not all tasks. Verbal tasks rely almost exclusively on the target form and PERMUTE has little to no effect. Verbal morphology is most often marked on the verb form itself, so this not surprising. Nouns and proper nouns behave similarly with the exception of case tasks. Case tasks show a mixed picture for all 4 parts of speech. TARG and PERMUTE both have a moderate effect. This might be explained by the fact that case is expressed in two distinct ways depending on the language. Agglutinative languages express case through suffixes, while analytic languages, such as English, express case with prepositions. In other words, the context is unnecessary for the first group and indispensable for the second.

Both TARG and PERMUTE are markedly small for gender and number tasks in adjectives. This is likely due to the fact that adjectives do not determine the gender or the number of the nominal head but rather copy (agree with) it.

Figure 8.5 shows the effect of TARG and PERMUTE by language family. Although the standard deviations are often larger than the mean effects, the trends are clear for multiple language families. The Uralic family is barely affected by PERMUTE while TARG has over 20% effect for both models. TARG has a larger effect than PERMUTE for the Baltic and the Romance family and isolate languages. Indic tasks on the other hand tend to have little change due to TARG, while PERMUTE has the largest effect for this family.

8.1.1.3 Relationship between perturbations

In the previous section we showed that TARG and PERMUTE often have an inverse correlation. Here we quantify their relationship as well as the relationship between all perturbations across the two models. First, we show that the effects across models are highly correlated as evidenced by Figure 8.6, which shows the pairwise Pearson correlation of the effects of each perturbation pair. The matrix is almost symmetrical. The main diagonal is close to one, which means that the same perturbation affects the two models in a very similar way. This suggests not just that the models are quite similar (see also Figure 8.7 depicting the correlation between perturbations in each model side by side) but also that the perturbations tell us more about morphology than about the models themselves.

8.1.2 Typology

While our dataset is too small for drawing far-reaching conclusions, we are beginning to see an emerging typological pattern in the effects of perturbation as defined in Equation 8.1. We cluster the languages by the effects of the perturbations on each task. There are 5 perturbations and 14 tasks, available as input features for the clustering algorithm, but many are missing in most languages. We use the column averages as imputation values. Since a single clustering run shows highly unstable results, we aggregate over 100 runs of K-means clustering with K drawn uniformly between 3 and 8 clusters. We then count how many times each pair of languages were clustered into the same cluster. Figure 8.8 illustrates the co-occurrence counts for XLM-RoBERTa. Since mBERT results are very similar, we limit our analysis to XLM-RoBERTa for simplicity.

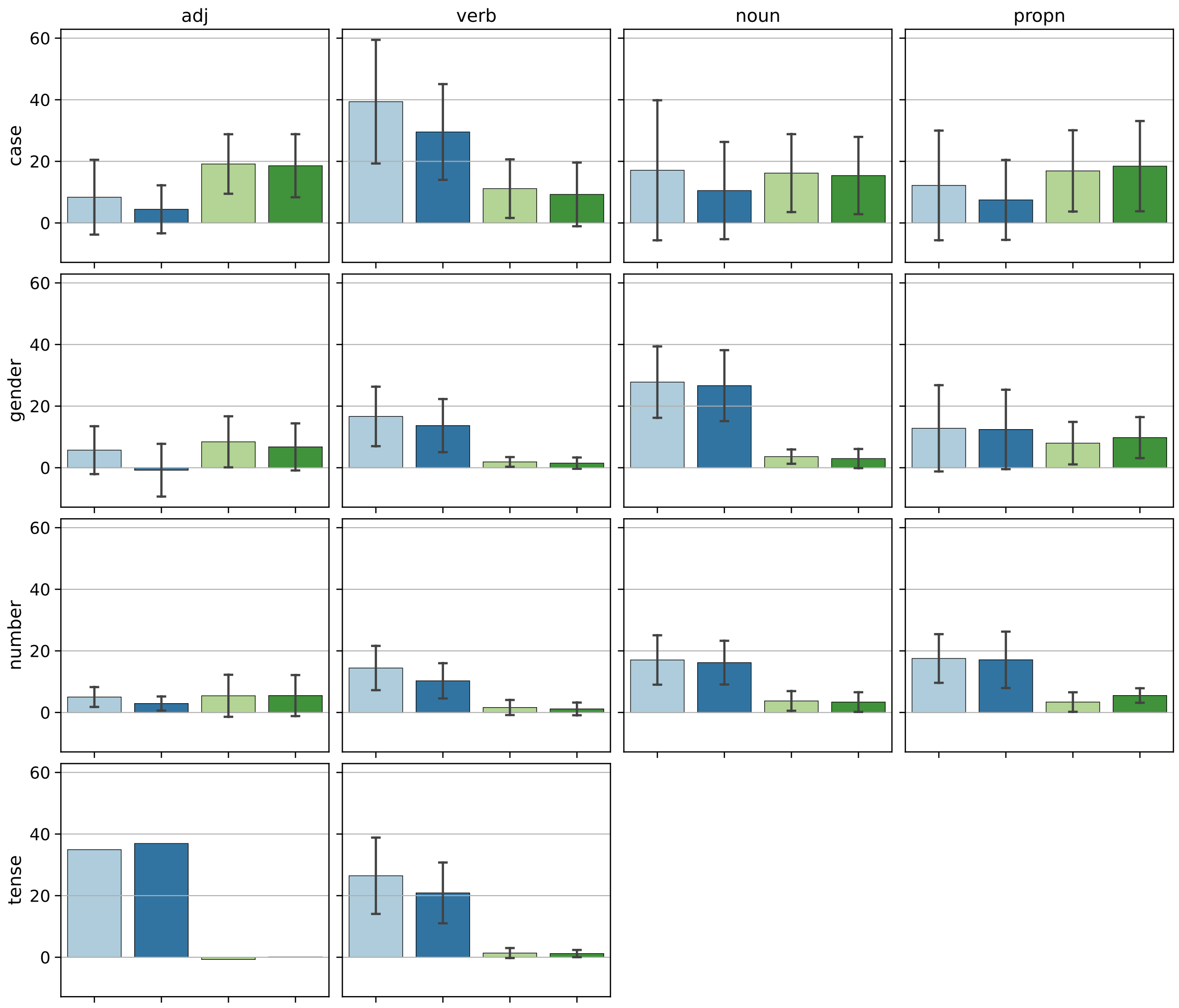

Figure 8.4: The effect of TARG and PERMUTE. Error bars indicate the standard deviation.

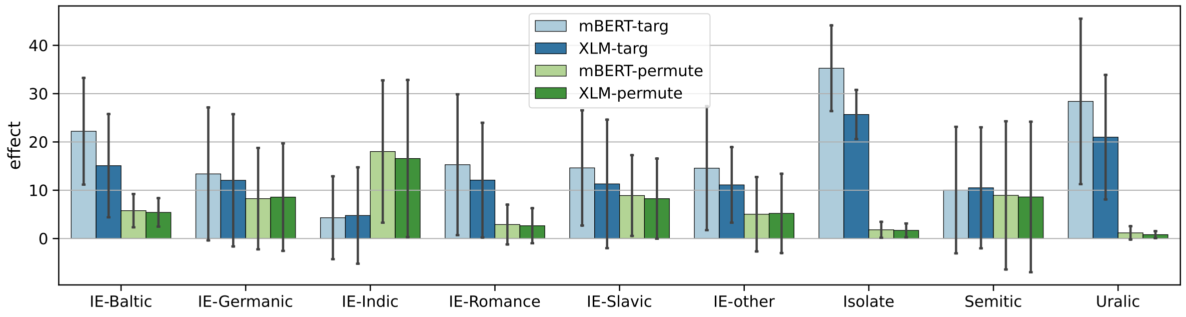

Figure 8.5: The effect of TARG and PERMUTE by language family. Error bars indicate the standard deviation.

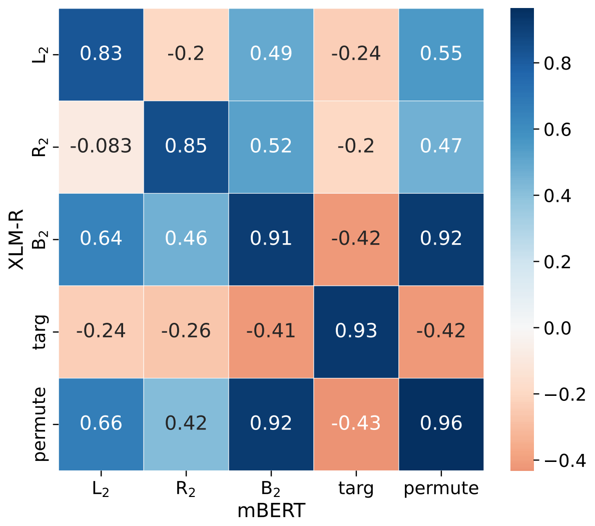

Figure 8.6: The pairwise Pearson correlation of perturbation effects between the two models.

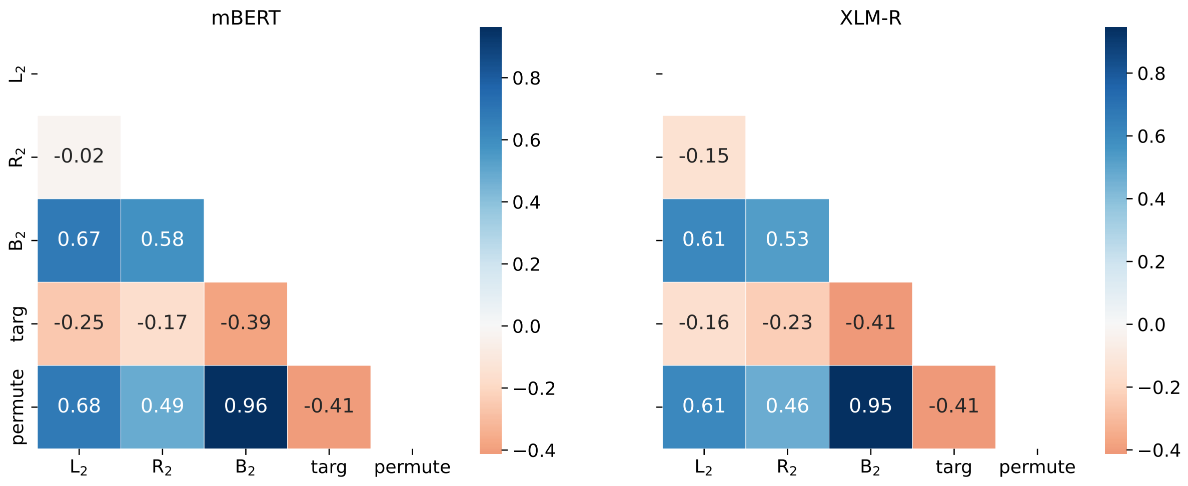

Figure 8.7: The pairwise Pearson correlation of perturbation effects by model.

Language families tend to be clustered together with some notable exceptions. German is seldom clustered together with other languages, including other members of the Germanic family. To a lesser extent, Latin is an outlier in the Romance family – it clusters better with Romanian than with Western or Southern Romance. The two Indic languages are almost always in a single cluster without any other languages, but the two Semitic languages are almost never in the same cluster. Arabic tends to be in its own cluster, while Hebrew is often grouped with Indo European languages. The Uralic family forms a strong cluster along with Basque and Turkish. These languages have highly complex agglutination and they all lack gender, so this is not surprising.

8.2 Shapley values

Having measured the (generally harmful) effect of perturbations, our next goal is to assign responsibility (blame) to the contributing factors. We use Shapley values for this purpose. For a general intro-

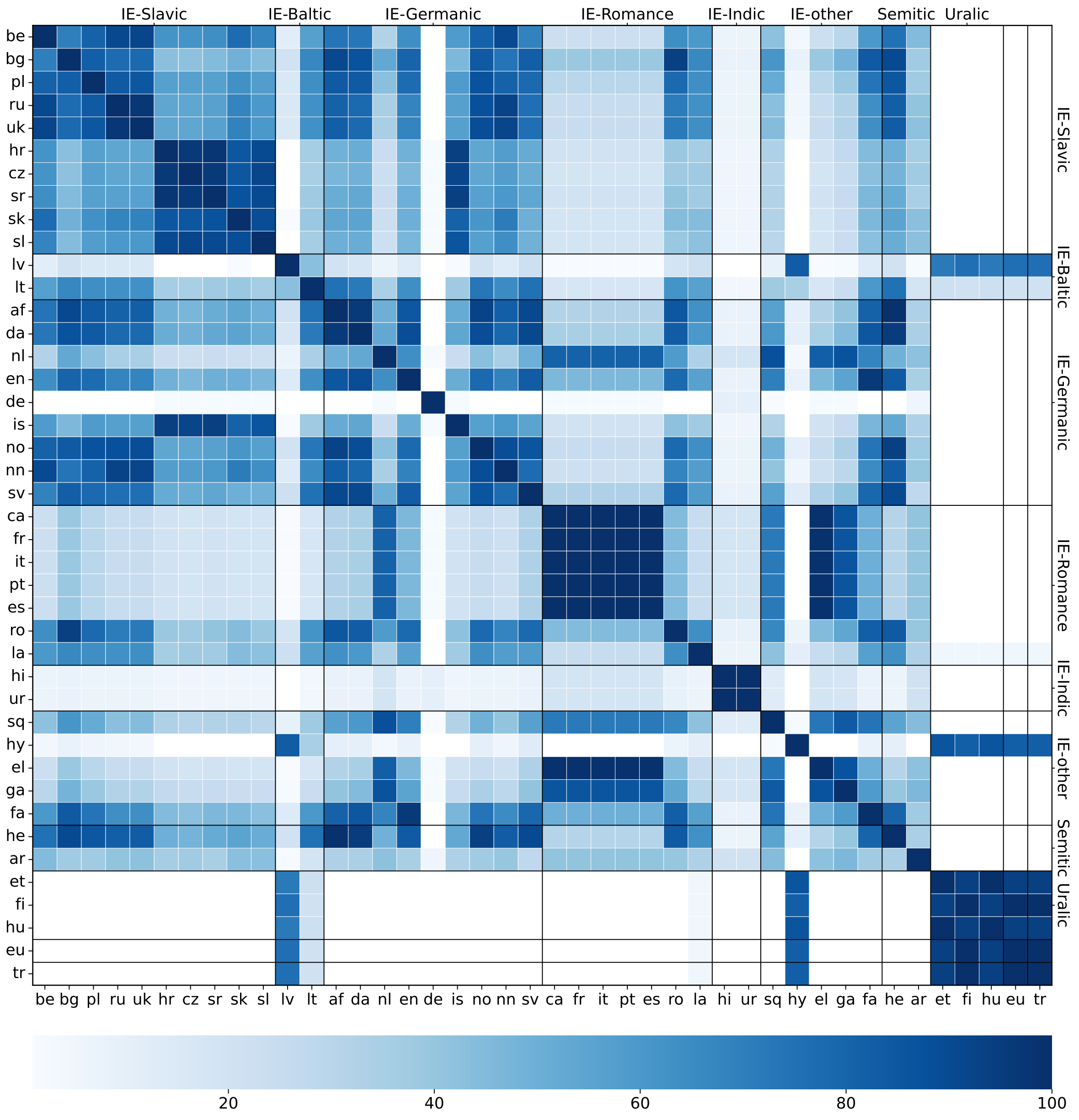

Figure 8.8: Co-occurrence counts for each languages pair over 100 clustering runs. Languages are sorted by family and a line is added between families.

duction, see Shapley (1951) and Lundberg and Lee (2017); for motivation of Shapley values in NLP, see Ethayarajh and Jurafsky (2021). We consider a probe as a coalition game of the words of the sentence. We treat each token position as a player in the game. The tokens are defined by their relative position to the target token. A sentence is a sequence defined as $L_k, L_{k-1}, \dots, L_1, T, R_1, R_2, \dots, R_m$, where k is the number of words that precede that target word and m is the number of words that follow it. The tokens far to the left are considered as belonging to a single position (-4^-), those far to the right to another position (4^+), so we have a total of 9 players $N = \{-4^-, -3, -2, -1, 0, 1, 2, 3, 4^+\}$.

On a given task, we can remove the contribution of a player i by masking the word(s) in positions corresponding to that player. The Shapley value $\varphi(i)$ corresponding to this player is computed as

$$\varphi(i) = \frac{1}{n} \sum_{S \subseteq N \setminus \{i\}} \frac{v(S \cup \{i\}) - v(S)}{\binom{n-1}{|S|}}, \tag{8.2}$$

where n is the total number of players, 9 in our case, and $v(S)$ is the value of coalition S (a set of players, here positions) on the given tasks. $v(S)$ is a function of the accuracies (Acc) of the task's probe with coalition S, the full set of players N, and the model. When all players are absent (masked), $\text{Acc}_{\text{all masked}}$ is very close to the accuracy of the trivial classifier that always picks the most common label. As is clear from Eq. 8.2, the contribution of the ith player is established as a weighted sum of the difference in the contributions of each coalition that contains i versus having i excluded. The weights are chosen to guarantee that these contributions are always additive: bringing players i and j into a coalition improves it exactly by $\varphi(i) + \varphi(j)$. The value of the entire set of players is always 1 (we use a multiplier 100 to report results in percentages), and we scale the contributions so that the value of the empty coalition is 0:

$$v(S) = 100 - 100 \cdot \frac{\text{Acc}_S - \text{Acc}_{\text{all masked}}}{\text{Acc}_{\text{mBERT}} - \text{Acc}_{\text{all masked}}}. \tag{8.3}$$

Not only are the Shapley values defined by Equation 8.2 an additive measure of the contributions that a particular player (in our case, the average word occurring in that position) makes to solving the task, but they define the only such measure (Shapley, 1951).

8.2.1 Implementation

Both mBERT and XLM-RoBERTa have built-in mask tokens that are used for the masked language modeling objective. We remove the contribution of certain tokens by replacing them with mask symbols. Multiple tokens can be removed at a time and we use a single mask token in place of each token. We designate an unused character as mask for the chLSTM experiments. When a token is masked, we replace each of its characters with this mask token. Computing the Shapley values for 9 players requires $2^9 = 512$ experiments for each of the 247 tasks. This includes the unmasked sentence (all players contribute) and the completely masked sentence (no players), where each token is replaced with a mask symbol.

The overall number of experiments we ran is 460k. The average runtime of an experiment is 7 seconds and the total runtime is roughly 38 days on a single GPU. We used a GeForce RTX 2080 Ti (12GB) and a Tesla V100 (16GB). The maximum number of epochs was set to 200 but in practice this is only reached in 2% of the experiments. Early stopping based on the development loss ends the experiments after 22 epochs on average.

model	left	right	target	context	left/right	left/context	right/context
mBERT	24.3	16.7	58.9	41.1	1.455	0.591	0.406
XLM-R	27.0	18.6	54.4	45.6	1.452	0.592	0.408
chLSTM	15.7	5.3	79.0	21.0	2.962	0.748	0.252

Table 8.3: Summary of the Shapley values.

8.2.2 General results

Figure 8.11 shows the Shapley values averaged over the 247 tasks for each model. Table 8.3 summarizes the numerical results. The values extracted from the two MLMs are remarkably similar. We quantify this similarity using L_1 (Manhattan) distance, which is 0.09 between the average Shapley values of all tasks.[4] The Shapley distributions obtained by XLM-RoBERTa and mBERT move closely together: the mean distance between Shapley values obtained from XLM-RoBERTa and mBERT is just 0.206, and of the 247 pairwise comparisons only 5 are more than two standard deviations above the mean. This means that in general Shapley values are more specific to the morphology of the language than to the model we probe. To simplify our analysis, we only discuss the XLM-RoBERTa results in detail since they show the same tendencies and are slightly better than the results achieved with mBERT.

The first observation is that the majority of the information, 54.9%, comes from the target words themselves, with the context contributing on average only 45.1%. This is by no means uniform over the different POS and tags. As illustrated by Figure 8.9 adjectives and proper nouns rely on the context more than common nouns or verbs. We see similar trends when we look at the per-tag Shapley values in Figure 8.10. Tense, an exclusively verbal tag, has the largest Shapley value for target out of the four tags. chLSTM, our non-pretrained baseline, while more reliant on the target for every category, shows the same trends, indicating that the explanation is, at least partially, linguistic.

Next, we observe that words farther away from the target contribute less, providing a window weighting scheme (kernel density function) broadly analogous to the windowing schemes used in speech processing (Harris, 1978).

Third, the low Shapley values at the two ends, summing to 11.2% in XLM-RoBERTa (11.0% in mBERT) go some way toward vindicating the standard practice in KWIC indexing (Luhn, 1959), which is to retain only three words on each side of the target. While the observation that this much context is sufficient for most purposes, including disambiguation and machine translation, goes back to the very beginnings of information retrieval (IR) and machine translation (MT) (Choueka and Lusignan, 1985), our findings provide the first quantifiable statement to this effect in MLMs (for HMMs, see Sharan et al., 2018) and open the way for further systematic study directly on IR and MT downstream tasks.

With this we are coming to our central observation, evident both from Figure 8.11 and from numerical considerations (Table 8.3): the decline is noticeably faster to the right than to the left, in spite of the fact that there is nothing in the model architecture to cause such an asymmetry. What is more, not even our experiments with random weighted MLMs (presented in 6.6.4) show such asymmetry.

Whatever happens before a target word is about 40% more relevant than whatever happens after it. In morphophonology ‘assimilation’ is standardly classified, depending on the direction of influence

[4]The Kullback-Leibler (KL) divergence is 0.014 bits, also very small, but we use L_1 in these comparisons, since individual Shapley values can be negative. L_2 (Euclidean) distance values would be just as good (the Pearson correlation between L_1 and L_2 is 0.983), but since Shapley values sum to 1 L_1 is easier to interpret. In what follows, “distance” always refers to L_1 distance.

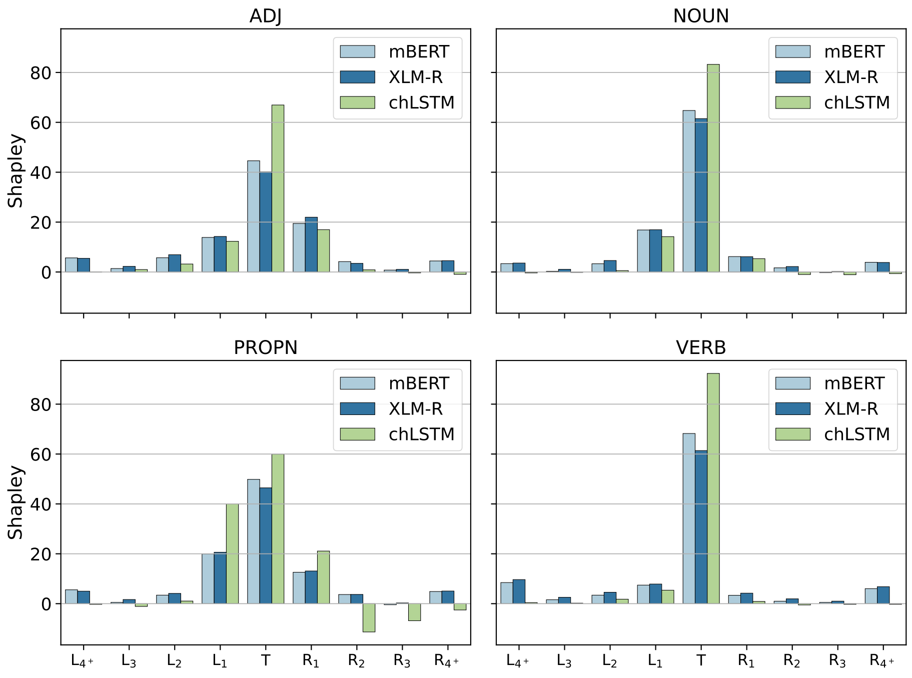

Figure 8.9: Shapley values by POS and model.

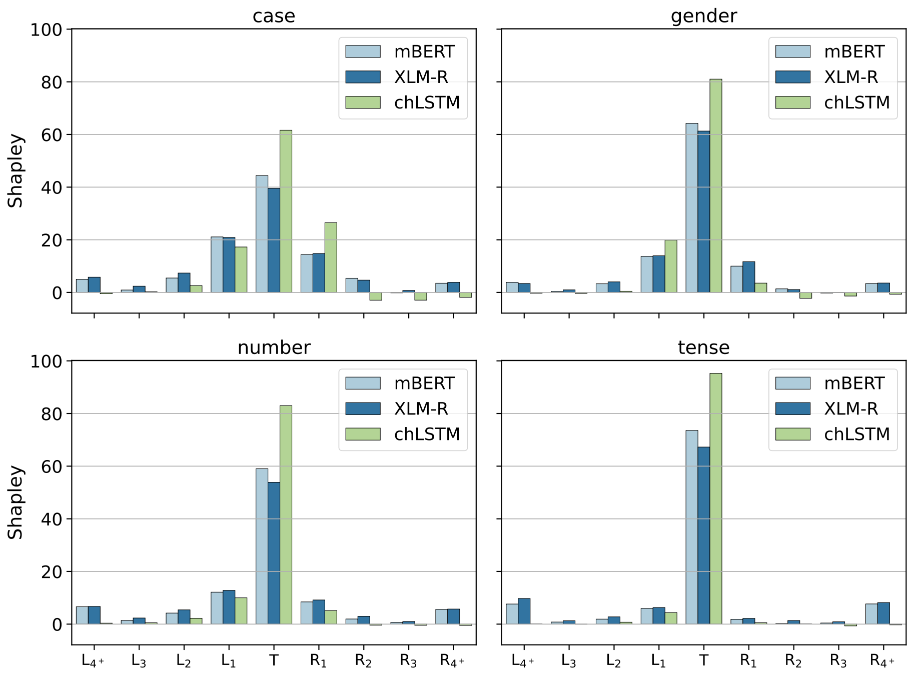

Figure 8.10: Shapley values by morphosyntactic tag and model.

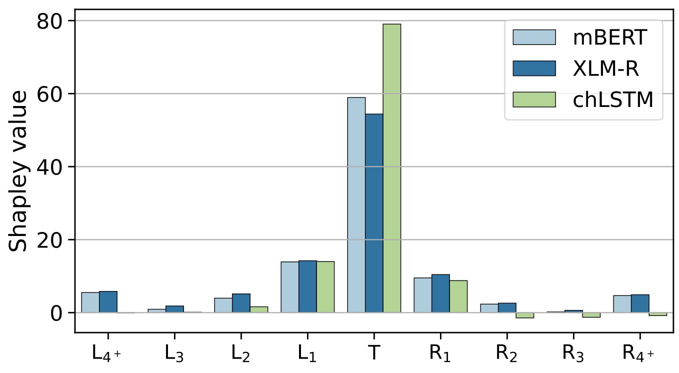

Figure 8.11: Shapley values by relative position to the probed target word. The values are averaged over the 247 tasks.

in a sequence, as *progressive* assimilation, in which a following element adapts itself to a preceding one, and *regressive* (or anticipatory) assimilation, in which a preceding element takes on a feature or features of a following one. What the Shapley values suggest for morphology is that progressive assimilation (feature spreading) is more relevant than regressive.

This is not to say that regressive assimilation will be impossible, or even rare. One can perfectly well imagine a language where adjectives precede the noun they modify and agree to them in gender:[5] this form of agreement is clearly anticipatory. Also, the direction of the spreading may depend more on structural position than linear order, cf. for example the 'head marking' versus 'dependent marking' distinction drawn by Nichols (1986). But when all is said and done, the Shapley values, having been obtained from models that are perfectly directionless, speak for themselves: left context dominates right 58.39% to 41.61% in XLM-RoBERTa (58.36% to 41.64% in mBERT) when context weights are considered 100%. This makes clear that it is progressive, rather than anticipatory, feature sharing that is the unmarked case. While our dataset is currently heavily skewed toward IE languages, so the result may not hold on a typologically more balanced sample, it is worth noting that the IE family is very broad typologically, and three of the four heaviest outliers (Hindi, Urdu, Irish) are from IE, only Arabic is not.

8.2.3 Outliers

We next consider the outliers. The main outliers are listed in Figure 8.12. We compute the distance of each task's Shapley values from the mean. Over 91.5% of the tasks are very close (Manhattan distance below one standard deviation, 0.264 to the mean of the distribution, and there are only 5 tasks (2% of the total) where the distance exceeds two standard deviations above the mean. The first row of Figure 8.12 shows the mean distribution and the 5 tasks that are *closest* to it, such as ⟨Polish, N, number⟩ (1st row 2nd panel, distance from mean (DFM) 0.053) or ⟨Lithuanian, N, case⟩ (1st row 3rd panel, DFM $= 0.132$). These exemplify the typologically least marked, simplest cases, and thus require no special explanation.

What does require explanation are the outliers, Shapley patterns far away from the norm. By distance from the mean, the biggest outliers are tasks from the Indic family: ⟨Hindi, PROPN, case⟩, ⟨Hindi, ADJ, case⟩, ⟨Urdu, NOUN, case⟩ and ⟨Urdu, PROPN, case⟩. For these proper noun and noun

[5]Indeed, there are several such languages in our sample such as German and most Slavic languages.

tasks, the greatest Shapley contribution, about 72–73%, is on the word following the proper noun. In Hindi not knowing the target is actually better than knowing it, the target's own contribution is negative 12% and its contribution is close to 0 in Urdu. For the case marked on Hindi adjectives (⟨Hindi, ADJ, case⟩), the most important is the second to its right, 59%; followed by the 3rd to the right, 15%; the target itself, 13%; and the first to the right, 12%. The Indic noun case patterns, unsurprisingly, follow closely the proper noun patterns. For both Hindi and Urdu there are good typological reasons such as SOV word order, for this to be so.[6] A closer look at all tasks from the Indic family (Figure 8.13) shows that only half of them are dominated by the target Shapley value.

The next biggest outliers are ⟨Arabic, NOUN, case⟩ and ⟨Irish, NOUN, case⟩, where the preceding word is more informative than the target itself. These are similarly explainable, this time by VSO order. It also stands to reason that the preceding word, typically an article, will be more informative about ⟨German, NOUN, case⟩ than the word itself. The same can be said about ⟨German, ADJ, gender⟩ and ⟨German, ADJ, number⟩, or the fact that ⟨Czech, ADJ, gender⟩ is determined by the following word, generally the head noun.

If we arrange Shapley distributions by decreasing distance from the mean, we see that dfm is roughly normally distributed (mean 0.492, std 0.264). Only 21 tasks are more than one standard deviation above the mean, the last two rows of Figure 8.12, present the top 12 of these. Altogether, there was a single case where R_2 dominated, ⟨Hindi, ADJ, case⟩, 16 cases when L_1 dominates, and 11 cases where R_1 dominates, everywhere else it is the target that is the most informative. The typologically unusual patterns, all clearly related to the grammar of the language in question, are transparently depicted in the Shapley patterns. For example, as noted in Section 6.5.5, the article preceding the noun in German often is the only indication of the noun's case. The Shapley values we obtained simply quantify this information dependence. Similarly, Arabic cases are determined in part by the preceding verb and/or preposition. Quite often, Shapley values confirm what we know anyway, e.g., that verbal tasks rely more on the target word than nominal tasks.

To the extent that similar rule-based explanations can be ascertained for all cases listed in Figure 8.12, we can attribute XLM-RoBERTa's success to an impressive sensitivity to grammatical regularities. Though the mechanisms are clearly different, such a finding places XLM-RoBERTa in the same broad tradition as other works seeking to discover rules and constraints (e.g., Brill, 1993).

Finally, we opted to mention the curious case of German tasks depicted in Figure 8.14. German is clearly the biggest outlier from the Germanic family. In fact, it is dissimilar to every other language as illustrated by our clustering experiments (cf. Figure 8.8). Multiple German tasks appear in the main Shapley outliers (Figure 8.12) and among the tasks with the least uniform layer weights (Figure 6.30). When looking at the individual tasks' Shapley values, only 4 of the 10 German tasks are dominated by the target. Case tasks are left dominant, while adjective gender and number rely on R_1 the most.

8.2.4 The difficulties of generalization

Our Shapley data can be summarized as a 9-dimensional vector for each ⟨i, j, k⟩. In other words, the Shapley distributions come naturally arranged in a 3D tensor. Unfortunately, many of the values are missing, either because the language does not combine a particular POS with a particular tag, or because we do not have enough data for training on the task. With a much larger dataset (recall from Section 6.3.1) that we use essentially all currently available uniformly coded data) experimenting with 3D tensor decomposition techniques (Kolda and Bader, 2009) may make sense, but for now the outcome depends too much on the imputation method. That said, we have obtained one robust

[6]We thank Paul Kiparsky for pointing this out.

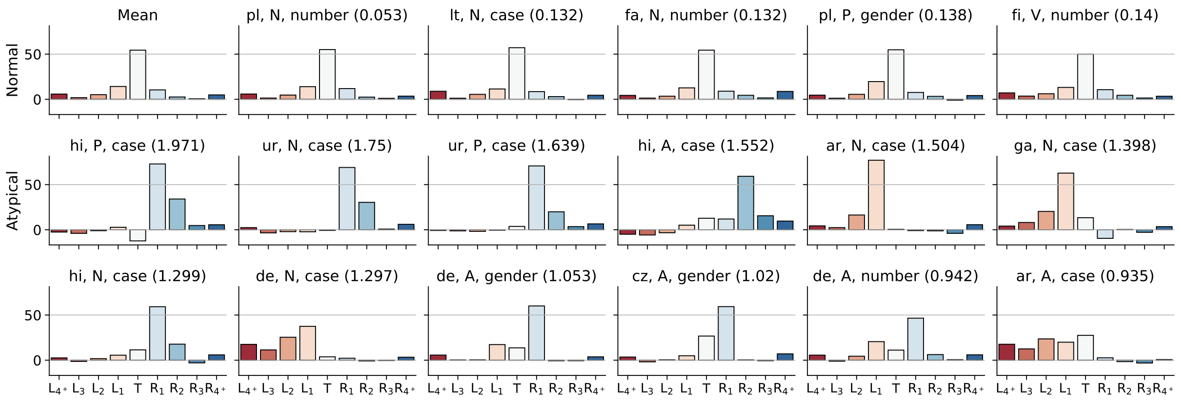

Figure 8.12: Least and most anomalous Shapley distributions. The first row are the mean Shapley values of the 247 tasks and the 5 tasks *closest* to the mean distribution, i.e. the least anomalous as measured by the dfm distance from the average Shapley values. The rest of the rows are the most anomalous Shapley values in descending order. For each particular task, its distance from the mean (dfm) is listed in parentheses above the graphs.

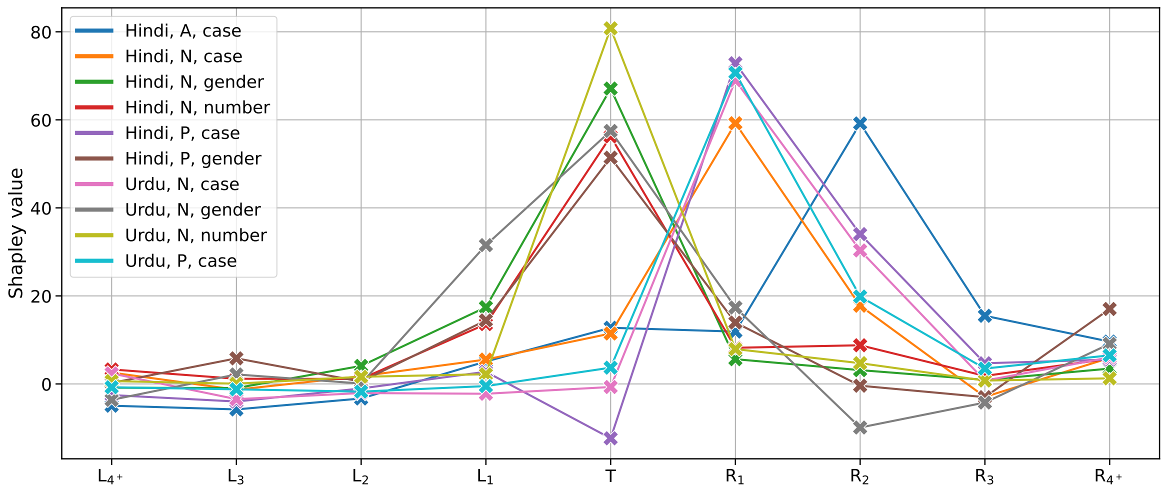

Figure 8.13: Shapley values in Indic tasks.

conclusion, independent of how we fill the missing values: it is harder to generalize by language than by POS or tag. It would be tempting to look at languages as units of generalization, but we found that trends rarely apply to individual languages!

For the totality of tasks ⟨i, j, k⟩ we can keep one of i, j, or k fixed, and compute the average Shapley distributions $S_{j,k}(i)$, $S_{i,k}(j)$, and $S_{i,j}(k)$. Given the average distributions, say $S_{j,k}$(Polish) we can ask how far Shapley distributions for all available Polish tasks are from it, and compute the average of these distances from the mean in the selected direction (in this case, language). We find that the average distance from the language averages is 0.37, while the distance from tag averages is 0.25 and the distance from POS averages is 0.26. In other words, aggregating tasks by language results in considerably larger variability than aggregating by POS or tag. The POS and tag results are similar since POS and tag are highly predictive of each other across languages: typically nouns will have case, verbs will have tense, and conversely, tenses are found on verbs, cases on nouns. This makes data aggregated on POS and tag jointly, as in Figure 8.4, much easier to make sense than data

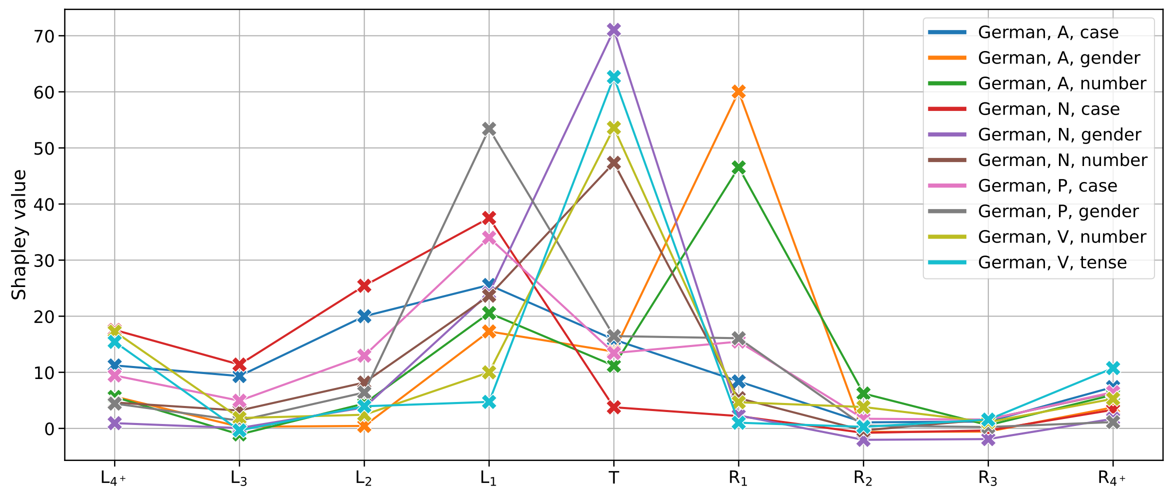

Figure 8.14: Shapley values in German tasks.

aggregated by language.

8.3 Conclusion

In this chapter we showed that perturbations or the systematic removal of information from a sentence and probing the incomplete sentence is a suitable tool for analyzing both the languages and the models. Since our conclusions show high agreement across models including a non-pretrained baseline, we attribute them to the languages rather than the models.

We further refined our analysis using Shapley values as a direct measure of token importance relative to the target token. We find general trends over the different tags, POS and language families, the most interesting outliers have linguistic explanations.

Finally we showed that generalizing over languages is difficult and it offers less of an explanation than generalizing over tags and POS.

Conclusion and Future Outlook

Summary of key contributions

The first part of my dissertation examines pre-Transformer deep learning models for morphology. These models are smaller and most often trained solely on the particular task without any pre-training. I showed that low level morphosyntactic tasks such as morphological inflection, analysis and lemmatization can be tackled with smaller deep learning models with high level accuracy. I compared different versions of encoder-decoder models with attention (Thesis 1). The most unique model is SoPa, a restricted finite state model implemented as a fully trainable neural network. Schwartz et al. (2018) created SoPa and applied it on sequence classification tasks which I extended with a decoder and used it on sequence-to-sequence tasks such as morphological analysis and lemmatization. Although the results were below the LSTM baselines, the additional interpretability made these experiments worthwhile (Thesis 2).

I turned towards pre-trained language models (PLMs) in the second part of this dissertation. PLMs revolutionized the field and are now omnipresent in our daily lives. My main research goal was to evaluate how good they are for morphology. I examined this question for 247 languages from 42 language families via probing and I showed that the multilingual models from BERT model family are indeed very good at morphology if we choose the subword pooling method carefully (Thesis 3).

I then turned to monolingual models for Uralic languages and compared them against multilingual models. I showed that monolingual models are usually better but the difference is small. Additionally I showed that BERT models can easily be transferred to unsupported languages for POS and NER tagging with modest amounts of annotated data (Thesis 4).

Finally I introduced a set of perturbations that remove certain sources of information from a sentence and I showed that probing performance is affected in ways that we often anticipate based on linguistic properties particularly the role of context. To further analyze this I computed the Shapley values of each token in the context. Aside from general trends, I often found linguistic explanation for the main outliers (Thesis 5).

Limitations and challenges

Part I. of this dissertation dates back to the time of smaller task-specific neural models which are less widely used nowadays especially for building new applications. The limitations at the time were mainly the availability of annotated training data or a rule-based morphological analyzer.

The main limitations of Part II. pertain to the nature of the tasks I examined. Morphosyntactic tasks are low-level tasks and the performance of certain models on these tasks may not give a lot of insight about what we can expect on high level tasks which are generally accepted as benchmarks when available. Unfortunately the large majority of languages, even ones with sufficient amounts of raw text data for LLM training, lack such benchmarks and I had to work with what was available. UD is one of the few standardized resources available in a large number of languages but it still only available in less than 10% of all languages. Thanks to the standard format of UD, I was able to do cross-lingual comparisons.

The second important limitation of Part II. is data quality. One source of data errors is UD itself. Although generally a high quality resource, it is composed of diverse treebanks, even within the same language, and it is bound to have errors and inconsistencies. The high number of languages I studied made it impossible to do specialized quality control. The second source of errors is the result of the sampling method itself. It considers every token as a candidate for target token but many tokens are parts of multiword expressions.

The third major limitation of Part II. is the lack of fine-tuning aside from a small ablation study (Section 6.6.3) and the evaluation of Uralic languages (Section 7.2). The main reason for this is efficiency. Fine-tuning would have resulted in an 80-fold increase in training time and an even larger increase in evaluation time due to the necessity of reloading the fine-tuned BERT models each time. Other technical difficulties of fine-tuning include a more sensitive batch size choice due to VRAM limitations and a much higher need for disk space. A case can be made for feature extraction as a better estimation of model knowledge than full scale fine-tuning.

Future research directions

The research questions I examined in Part I. are becoming less and less relevant nowadays since LLMs seem to pick up morphosyntax during training and high-level problems such as hallucination are more crucial. There is a sharp decline in morphology-related papers at major conferences and the SIGMORPHON workshop is unfortunately getting smaller every year. Still, morphosyntax remains relevant for low-resource languages where data collection is often hindered by the lack of human resources and high quality morphological analyzers can help linguists in data curation.

My studies in Part II. mainly targeted models with hundreds of millions of parameters rather than billion parameter LLMs popular today. Aside from the fact that most of those experiments predate newer LLMs, medium sized models are easier to study, particularly when fine-tuning is involved. An obvious extension of my work is the probing of large open-source LLMs. Some LLMs such as the GPT-3 and GPT-4 family are only available through APIs which makes probing impossible due to the inaccessibility of output tensors. Prompting can be used as a workaround but it has its own challenges as I pointed out in Section 6.5.6. An unexplored future direction of my work is improving the prompts.

Another angle to explore is the few shot training of LLMs before prompting or probing. LLMs are famously good at picking up new tasks after just a few well chosen training samples. The difficulty lies in the choice of training samples in this case. Since morphosyntactic probing is generally easy for both PLMs and the best baselines, the choice of those few samples has to be made carefully to include both typical and challenging cases.

Finally, expanding the dataset is a straightforward continuation of my work. I artificially limited the selection of morphosyntactic tags to 4 common tags in Chapter 6 to make cross-lingual comparisons possible but using a wider selection of tags brings additional insight as shown in Chapter 7. The choice of languages was limited by both UD and the models we studied but UD has expanded since then and the models are often applicable to unsupported languages with some fine-tuning as shown in Section 7.2.

Final thoughts

As deep learning models were gaining popularity around 2014, I set out to prove that rule-based computational morphology still had its hard-earned place in NLP. NLP conferences were getting bigger every year and the sheer amount of high quality papers keeps growing to this day. Morphology, due to its minimal role in English, always constituted a niche field. Still, I believe that although modern LLMs handle morphology with near perfect accuracy, the remaining issues are interesting and we can derive useful information from these particularly in a multilingual setting.

Publications related to this dissertation

Judit Ács. 2018. BME-HAS system for CoNLL–SIGMORPHON 2018 shared task: Universal morphological reinflection. *Proceedings of the CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 121–126.

Judit Ács. 2019. Exploring BERT's vocabulary. http://juditacs.github.io/2019/02/19/bert-tokenization-stats.html. Accessed: 2021-05-14.

Judit Ács, Endre Hamerlik, Roy Schwartz, Noah A. Smith, and Andras Kornai. 2023. Morphosyntactic probing of multilingual BERT models. *Natural Language Engineering*, page 1–40.

Judit Ács, Ákos Kádár, and Andras Kornai. 2021a. Subword pooling makes a difference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2284–2295, Online. Association for Computational Linguistics.

Judit Ács and András Kornai. 2020. The role of interpretable patterns in deep learning for morphology. In *XVI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2020)*, pages 171–179, Szeged.

Judit Ács, Dániel Lévai, and Andras Kornai. 2021b. Evaluating transferability of BERT models on Uralic languages. In *Proceedings of the Seventh International Workshop on Computational Linguistics of Uralic Languages*, pages 8–17, Syktyvkar, Russia (Online). Association for Computational Linguistics.

Judit Ács, Dániel Lévai, Dávid Márk Nemeskey, and András Kornai. 2021. Evaluating contextualized language models for Hungarian. In *XVII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2020)*, Szeged.

Ádám Kovács, Judit Ács, András Kornai, and Gábor Recski. 2020. Better together: Modern methods plus traditional thinking in NP alignment. In *Proc. LREC 2020*, pages 3635—-3639.

Ádám Kovács, Evelin Ács, Judit Ács, András Kornai, and Gábor Recski. 2019. BME-UW at SRST-2019: Surface realization with interpreted regular tree grammars. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation (MSR 2019)*, pages 35–40, Hong Kong, China. Association for Computational Linguistics.

Attila Nagy, Bence Bial, and Judit Ács. 2021. Automatic punctuation restoration with BERT models. In *XVIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2021)*.

Tiago Pimentel, Maria Ryskina, Sabrina J. Mielke, Shijie Wu, Eleanor Chodroff, Brian Leonard, Garrett Nicolai, Yustinus Ghanggo Ate, Salam Khalifa, Nizar Habash, Charbel El-Khaissi, Omer Goldman, Michael Gasser, William Lane, Matt Coler, Arturo Oncevay, Jaime Rafael Montoya Samame, Gema Celeste Silva Villegas, Adam Ek, Jean-Philippe Bernardy, Andrey Shcherbakov, Aziyana Bayyr-ool, Karina Sheifer, Sofya Ganieva, Matvey Plugaryov, Elena Klyachko, Ali Salehi, Andrew Krizhanovsky, Natalia Krizhanovsky, Clara Vania, Sardana Ivanova, Aelita Salchak, Christopher Straughn, Zoey Liu, Jonathan North Washington, Duygu Ataman, Witold Kieraś, Marcin Woliński, Totok Suhardijanto, Niklas Stoehr, Zahroh Nuriah, Shyam Ratan, Francis M. Tyers, Edoardo M. Ponti, Grant Aiton, Richard J. Hatcher, Emily Prud'hommeaux, Ritesh Kumar, Mans Hulden, Botond Barta, Dorina Lakatos, Gábor Szolnok, Judit Ács, Mohit Raj, David Yarowsky, Ryan Cotterell, Ben Ambridge, and Ekaterina Vylomova. 2021. SIGMORPHON 2021 shared task on morphological reinflection: Generalization across languages. In *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 229–259, Online. Association for Computational Linguistics.

Gábor Recski, Ádám Kovács, Kinga Gémes, Judit Ács, and Andras Kornai. 2020. BME-TUW at SR'20: Lexical grammar induction for surface realization. In *Proceedings of the Third Workshop on Multilingual Surface Realisation*, pages 21–29, Barcelona, Spain (Online). Association for Computational Linguistics.

Gábor Szolnok, Botond Barta, Dorina Lakatos, and Judit Ács. 2021. BME submission for SIGMORPHON 2021 shared task 0. A three step training approach with data augmentation for morphological inflection. In *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 268–273, Online. Association for Computational Linguistics.

Other publications

Judit Ács. 2014. Pivot-based multilingual dictionary building using Wiktionary. In *The 9th edition of the Language Resources and Evaluation Conference*.

Judit Ács. 2015. Synonym acquisition from translation graph. In *XI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2015)*, pages 14–21. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Judit Ács, Gábor Borbély, Márton Makrai, Dávid Nemeskey, Gábor Recski, and András Kornai. 2018. Hibrid nyelvtechnológiák. *Magyar Tudomány*, 6.

Judit Ács, László Grad-Gyenge, and Thiago Bruno Rodrigues de Rezende Oliveira. 2015. A two-level classifier for discriminating similar languages. In *Proceedings of the Joint Workshop on Language Technology for Closely Related Languages, Varieties and Dialects*, pages 73–77, Hissar, Bulgaria. Association for Computational Linguistics.

Judit Ács and József Halmi. 2016a. Comparing diacritic restoration methods for Hungarian. In *Proceedings of the Automation and Applied Computer Science Workshop 2016 : AACS'16*. Budapest University of Technology and Economics.

Judit Ács and József Halmi. 2016b. Hunaccent: Small footprint diacritic restoration for social media. In *Proceedings of the First Workshop on Normalisation and Analysis of Social Media Texts (NormSoMe)*.

Judit Ács and Ágota Illyés. 2015. Language detection and generation. In *Proceedings of the Automation and Applied Computer Science Workshop 2015 : AACS'15*. Budapest University of Technology and Economics.

Judit Ács and András Kornai. 2016. Evaluating embeddings on dictionary-based similarity. In *Proceedings of the First Workshop on Evaluating Vector-Space Representations for NLP (RepEval)*, pages 78–82.

Judit Ács, Dávid Nemeskey, and András Kornai. 2017. Identification of disaster-implicated named entities. In *Proceedings of the First International Workshop on Exploitation of Social Media for Emergency Relief and Preparedness*.

Judit Ács, Dávid Márk Nemeskey, and Gábor Recski. 2017. Building word embeddings from dictionary definitions. In Katalin Mády Beáta Gyuris and Gábor Recski, editors, *K + K = 120: Papers dedicated to László Kálmán and András Kornai on the occasion of their 60th birthdays*. Research Institute for Linguistics, Hungarian Academy of Sciences (RIL HAS).

Judit Ács, Katalin Pajkossy, and András Kornai. 2013. Building basic vocabulary across 40 languages. In *Proceedings of the Sixth Workshop on Building and Using Comparable Corpora*, pages 52–58, Sofia, Bulgaria. Association for Computational Linguistics.

Judit Ács and Géza Velkey. 2017. Comparing word segmentation algorithms. In *Proceedings of the Automation and Applied Computer Science Workshop 2017 : AACS'17*. Budapest University of Technology and Economics.

Botond Barta, Endre Hamerlik, Milán Konor Nyist, and Judit Ács. 2025. HuAMR: A Hungarian AMR parser and dataset. In *XXI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2025)*, pages 185–196. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Botond Barta, Dorina Lakatos, Attila Nagy, Milán Konor Nyist, and Judit Ács. 2023. HunSum-1: an abstractive summarization dataset for Hungarian. In *XIX. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2023)*, pages 231–243. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Botond Barta, Dorina Lakatos, Attila Nagy, Milán Konor Nyist, and Judit Ács. 2024. From news to summaries: Building a Hungarian corpus for extractive and abstractive summarization. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 7503–7509, Torino, Italia. ELRA and ICCL.

Dániel Huszti and Judit Ács. 2017. Entitásorientált véleménykinyerés magyar nyelven. In *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2017)*, pages 240–250. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

András Kornai, Judit Ács, Márton Makrai, Dávid Márk Nemeskey, Katalin Pajkossy, and Gábor Recski. 2015. Competence in lexical semantics. In *Proceedings of the Fourth Joint Conference on Lexical and Computational Semantics*, pages 165–175, Denver, Colorado. Association for Computational Linguistics.

Attila Nagy, Dorina Lakatos, Botond Barta, and Judit Ács. 2023a. TreeSwap: Data augmentation for machine translation via dependency subtree swapping. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pages 759–768, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria.

Attila Nagy, Dorina Petra Lakatos, Botond Barta, Patrick Nanys, and Judit Ács. 2023b. Data augmentation for machine translation via dependency subtree swapping. In *XIX. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2023)*. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Attila Nagy, Patrick Nanys, Konrád Balázs Frey, Bence Bial, and Judit Ács. 2022. Syntax-based data augmentation for Hungarian-English machine translation. In *XVIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2022)*. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Gergely Dániel Németh and Judit Ács. 2018. Hyphenation using deep neural networks. In *XIV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2018)*. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Gábor Recski and Judit Ács. 2015. MathLingBudapest: Concept networks for semantic similarity. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 138–142, Denver, Colorado. Association for Computational Linguistics.

Bibliography

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *Proceedings of International Conference on Learning Representations*.

Roee Aharoni and Yoav Goldberg. 2016. Morphological inflection generation with hard monotonic attention. *arXiv preprint arXiv:1611.01487*.

Roee Aharoni, Yoav Goldberg, and Yonatan Belinkov. 2016. Improving sequence to sequence learning for morphological inflection generation: The BIU-MIT systems for the SIGMORPHON 2016 shared task for morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 41–48, Berlin, Germany. Association for Computational Linguistics.

László Antal. 1963. The possessive form of the Hungarian noun. *Linguistics*, 3:50–61.

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. 2019. Tuning multilingual transformers for language-specific named entity recognition. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy. Association for Computational Linguistics.

Þórunn Arnardóttir, Hinrik Hafsteinsson, Einar Freyr Sigurðsson, Kristín Bjarnadóttir, Anton Karl Ingason, Hildur Jónsdóttir, and Steinþór Steingrímsson. 2020. A Universal Dependencies conversion pipeline for a Penn-format constituency treebank. In *Proceedings of the Fourth Workshop on Universal Dependencies (UDW 2020)*, pages 16–25, Barcelona, Spain (Online). Association for Computational Linguistics.

BIG bench authors. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR 2015)*.

Yonatan Belinkov. 2021. Probing Classifiers: Promises, Shortcomings, and Advances. *arXiv:2102.12452 [cs]*. ArXiv: 2102.12452.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017a. What do neural machine translation models learn about morphology? In *Proc. of ACL*.

Yonatan Belinkov, Lluís Màrquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. 2017b. Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. In *IJCNLP*. ArXiv preprint arXiv:1801.07772.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Fadi Biadsy, Nizar Habash, and Julia Hirschberg. 2009. Improving the Arabic pronunciation dictionary for phone and word recognition with linguistically-based pronunciation rules. In *Proceedings of human language technologies: The 2009 annual conference of NAACL*, pages 397–405.

Gábor Borbély, Márton Makrai, Dávid Márk Nemeskey, and András Kornai. 2016. Evaluating multi-sense embeddings for semantic resolution monolingually and in word translation. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 83–89. Association for Computational Linguistics.

Eric Brill. 1993. Automatic grammar induction and parsing free text: A transformation-based approach. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 259–265, Columbus, Ohio, USA. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jose Camacho-Collados and Mohammad Taher Pilehvar. 2018. From word to sense embeddings: A survey on vector representations of meaning. *J. Artif. Int. Res.*, 63(1):743–788.

Kris Cao and Marek Rei. 2016. A joint model for word embedding and word morphology. In *Repl4NLP*.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, 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.

Yaacov Choueka and Serge Lusignan. 1985. Disambiguation by short contexts. *Computers and the Humanities*, 19(3):147–157.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau, Germán Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. 2018a. What you can cram into a single \$&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. In *Advances in Neural Information Processing Systems*, pages 7057–7067.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2017. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. 2018b. XNLI: Evaluating cross-lingual sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Arya D. McCarthy, Katharina Kann, Sebastian Mielke, Garrett Nicolai, Miikka Silfverberg, David Yarowsky, Jason Eisner, and Mans Hulden. 2018. The CoNLL–SIGMORPHON 2018 shared task: Universal morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, Brussels, Belgium. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2017. The CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, Vancouver, Canada. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner, and Mans Hulden. 2016. The SIGMORPHON 2016 shared task—morphological reinflection. In *Proceedings of the 2016 Meeting of SIGMORPHON*, Berlin, Germany. Association for Computational Linguistics.

Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1287–1292.

Gözde Gül Şahin, Clara Vania, Ilia Kuznetsov, and Iryna Gurevych. 2019. LINSPECTOR: Multilingual probing tasks for word representations. arXiv:1903.09442.

Balázs Csanád Csáji. 2001. Approximation with Artificial Neural Networks. *MSc Thesis, Dept. Science, Eötvös Loránd University, Budapest, Hungary.*

Dóra Csendes, János Csirik, Tibor Gyimóthy, and András Kocsor. 2005. The Szeged Treebank. In *Lecture Notes in Computer Science: Text, Speech and Dialogue*, pages 123–131. Springer.

George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the*

North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning.

Philipp Dufter and Hinrich Schütze. 2020. Identifying elements essential for BERT's multilinguality. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4423–4437, Online. Association for Computational Linguistics.

Stefan Dumitrescu, Petru Rebeja, Beata Lorincz, Mihaela Gaman, Andrei Avram, Mihai Ilie, Andrei Pruteanu, Adriana Stan, Lorena Rosia, Cristina Iacobescu, Luciana Morogan, George Dima, Gabriel Marchidan, Traian Rebedea, Madalina Chitez, Dani Yogatama, Sebastian Ruder, Radu Tudor Ionescu, Razvan Pascanu, and Viorica Patraucean. 2021. LiRo: Benchmark and leaderboard for Romanian language tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Yanai Elazar, Shauli Ravfogel, Alon Jacovi, and Yoav Goldberg. 2020. Amnesic probing: Behavioral explanation with amnesic counterfactuals. *Transactions of the Association for Computational Linguistics*, 9:160–175.

Kawin Ethayarajh and Dan Jurafsky. 2021. Attention flows are shapley value explanations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 49–54, Online. Association for Computational Linguistics.

Allyson Ettinger. 2020. What bert is not: Lessons from a new suite of psycholinguistic diagnostics for language models. *Transactions of the Association for Computational Linguistics*, 8:34–48.

Richárd Farkas, Veronika Vincze, and Helmut Schmid. 2012. Dependency parsing of Hungarian: Baseline results and challenges. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 55–65, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38.

Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Abhijeet Gupta, Gemma Boleda, Marco Baroni, and Sebastian Padó. 2015. Distributional vectors encode referential attributes. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 12–21, Lisbon, Portugal. Association for Computational Linguistics.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman, and Noah A. Smith. 2018. Annotation artifacts in natural language inference data. 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 107–112, New Orleans, Louisiana. Association for Computational Linguistics.

Frederic J. Harris. 1978. On the use of windows for harmonic analysis with the dft. *Proc. IEEE*, 66(1):51–83.

Zellig S. Harris. 1954. Distributional structure. *Word*, 10(23):146–162.

Betty Hart and Todd R. Risley. 1995. *Meaningful Differences in the Everyday Experience of Young American Children*. Brookes Publishing Company, Inc.

John Hewitt, Kawin Ethayarajh, Percy Liang, and Christopher Manning. 2021. Conditional probing: measuring usable information beyond a baseline. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1626–1639, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. 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 2733–2743, Hong Kong, China. Association for Computational Linguistics.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. 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 4129–4138.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. 2017. What do we need to build explainable ai systems for the medical domain?

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R. Bowman. 2019. Do attention heads in BERT track syntactic dependencies?

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. XTREME: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization. *CoRR*, abs/2003.11080.

Balázs Indig, Bálint Sass, Eszter Simon, Iván Mittelholcz, Péter Kundráth, and Noémi Vadász. 2019. emtsv – Egy formátum mind felett [emtsv – One format to rule them all]. In *XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2019)*, pages 235–247. Szegedi Tudományegyetem Informatikai Tanszékcsoport.

Bushra Jawaid and Daniel Zeman. 2011. Word-order issues in English-to-Urdu statistical machine translation. *The Prague Bulletin of Mathematical Linguistics*, 95(95):87–106.

Katharina Kann, Ryan Cotterell, and Hinrich Schütze. 2016. Neural morphological analysis: Encoding-decoding canonical segments. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 961–967, Austin, Texas. Association for Computational Linguistics.

Katharina Kann and Hinrich Schütze. 2016a. MED: The LMU system for the SIGMORPHON 2016 shared task on morphological reinflection. *ACL 2016*, page 62.

Katharina Kann and Hinrich Schütze. 2016b. MED: The LMU system for the SIGMORPHON 2016 shared task on morphological reinflection. In Kann and Schütze (2016a), page 62.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-Thought Vectors. In *Neural Information Processing Systems*.

Christo Kirov, Ryan Cotterell, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sebastian Mielke, Arya D. McCarthy, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. 2018. Unimorph 2.0: Universal morphology. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Arne Köhn. 2015. What's in an embedding? analyzing word embeddings through multilingual evaluation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2067–2073, Lisbon, Portugal. Association for Computational Linguistics.

Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. 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 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Nelda Kote, Marenglen Biba, Jenna Kanerva, Samuel Rönnqvist, and Filip Ginter. 2019. Morphological tagging and lemmatization of Albanian: A manually annotated corpus and neural models.

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.

Yuri Kuratov and Mikhail Arkhipov. 2019. Adaptation of deep bidirectional multilingual transformers for russian language.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *Advances in Neural Information Processing Systems (NeurIPS)*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. 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 4582–4597, Online. Association for Computational Linguistics.

N. Ligeti-Nagy, G. Ferenczi, E. Héja, K. Jelencsik-Mátyus, L. J. Laki, N. Vadász, Z. Gy. Yang, and T. Váradi. 2022. Hulu: magyar nyelvű benchmark adatbázis kiépítése a neurális nyelvmodellek kiértékelése céljából. In *XVIII. Magyar Számítógépes Nyelvészeti Konferencia*, pages 431–446, Szeged. Szegedi Tudományegyetem, Informatikai Intézet.

Krister Lindén, Erik Axelson, Sam Hardwick, Flammie A. Pirinen, and Miikka Silfverberg. 2011. Hfst - framework for compiling and applying morphologies. In *International Workshop on Systems and Frameworks for Computational Morphology*.

Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters, and Noah A. Smith. 2019a. Linguistic knowledge and transferability of contextual representations. 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 1073–1094, Minneapolis, Minnesota. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. RoBERTa: A robustly optimized bert pretraining approach.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

H. P. Luhn. 1959. Keyword-in-Context Index for Technical Literature (KWIC Index). *American Documentation*, 11(4):288–295.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774. Curran Associates, Inc.

Siwen Luo, Hamish Ivison, Soyeon Caren Han, and Josiah Poon. 2021. Local interpretations for explainable natural language processing: A survey. *ACM Computing Surveys*, 56:1 – 36.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics.

Aniek F. Markus, Jan A. Kors, and Peter R. Rijnbeek. 2021. The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies. *Journal of Biomedical Informatics*, 113:103655.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.

W.S. McCulloch and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, 5:115–133.

Igor A. Mel'cuk. 1972. On the possessive forms of the Hungarian noun. In Ferenc Kiefer and N. Rouwet, editors, *Generative grammar in Europe*, pages 315–332. Reidel, Dordrecht.

Márton Miháltz. 2013. OpinHuBank: szabadon hozzáférhető annotált korpusz magyar nyelvű véleményelemzéshez. In *IX. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2013)*, pages 343–345, Szeged.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Marvin Minsky and Seymour Papert. 1969. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Dávid Márk Nemeskey. 2020a. *Natural Language Processing Methods for Language Modeling*. Ph.D. thesis, Eötvös Loránd University.

Dávid Márk Nemeskey. 2020b. Webcorpus 2.0.

Dávid Márk Nemeskey. 2021. Introducing `huBERT`. In *XVII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2021)*, pages 3–14, Szeged.

Johanna Nichols. 1986. Head-marking and dependent-marking grammar. *Language*, 62:56–119.

Joakim Nivre, Mitchell Abrams, Željko Agić, et al. 2018. Universal Dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Attila Novák, Péter Rebrus, and Zsófia Ludányi. 2017. Az `emMorph` morfológiai elemző annotációs formalizmusa. In *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*, Szeged.

Attila Novák, Borbála Siklósi, and Csaba Oravecz. 2016. A new integrated open-source morphological analyzer for Hungarian. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

OpenAI. 2023. Gpt-4 technical report.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. 2017. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada. Association for Computational Linguistics.

Sungjoon Park, Jihyung Moon, Sungdong Kim, Won Ik Cho, Jiyoon Han, Jangwon Park, Chisung Song, Junseong Kim, Yongsook Song, Taehwan Oh, Joohong Lee, Juhyun Oh, Sungwon Lyu, Younghoon Jeong, Inkwon Lee, Sangwoo Seo, Dongjun Lee, Hyunwoo Kim, Myeonghwa Lee, Seongbo Jang, Seungwon Do, Sunkyoung Kim, Kyungtae Lim, Jongwon Lee, Kyumin Park, Jamin Shin, Seonghyun Kim, Lucy Park, Alice Oh, Jungwoo Ha, and Kyunghyun Cho. 2021. Klue: Korean language understanding evaluation.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.

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. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. https://github.com/openai/gpt-2.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Lance Ramshaw and Mitch Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Cambridge, MA. SIGDAT/ACL.

Abhilasha Ravichander, Yonatan Belinkov, and Eduard Hovy. 2021. Probing the probing paradigm: Does probing accuracy entail task relevance? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3363–3377, Online. Association for Computational Linguistics.

Péter Rebrus, András Kornai, and Dániel Varga. 2012. Egy általános célú morfológiai annotáció. *Általános Nyelvészeti Tanulmányok*, 24.

Frank Rosenblatt. 1957. The perceptron: a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory.

Dana Rubinstein, Effi Levi, Roy Schwartz, and Ari Rappoport. 2015. How well do distributional models capture different types of semantic knowledge? In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 726–730, Beijing, China. Association for Computational Linguistics.

Bálint Sass, Márton Miháltz, and Péter Kundráth. 2017. Az `e-magyar` rendszer gate környezetbe integrált magyar szövegfeldolgozó eszközlánca. In *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*, page (this volume), Szeged.

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. IEEE.

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. SoPa: Bridging CNNs, RNNs, and Weighted Finite-State Machines. In *Proc. 56th ACL Annual Meeting*, pages 295–305, Melbourne, Australia.

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 Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Lloyd S Shapley. 1951. Notes on the n-person game – ii: The value of an n-person game. Technical report, RAND Corporation.

Vatsal Sharan, Sam Khakade, Percy Liang, and Gregory Valiant. 2018. Prediction with a short memory. In *Proc. STOC 2018*, pages 1074–1087.

Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. 2018. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural MT learn source syntax? In *Proc. of EMNLP*.

Eszter Simon and Noémi Vadász. 2021. Introducing nytk-nerkor, A gold standard hungarian named entity annotated corpus. In *Text, Speech, and Dialogue - 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6-9, 2021, Proceedings*, volume 12848 of *Lecture Notes in Computer Science*, pages 222–234. Springer.

Koustuv Sinha, Prasanna Parthasarathi, Joelle Pineau, and Adina Williams. 2021. UnNatural Language Inference. 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 7329–7346, Online. Association for Computational Linguistics.

Péter Siptár and Miklós Törkenczy. 2007. *The Phonology of Hungarian.* Oxford University Press.

Hedvig Skirgård, Hannah J. Haynie, Damián E. Blasi, Harald Hammarström, Jeremy Collins, Jay J. Latarche, Jakob Lesage, Tobias Weber, Alena Witzlack-Makarevich, Sam Passmore, Angela Chira, Luke Maurits, Russell Dinnage, Michael Dunn, Ger Reesink, Ruth Singer, Claire Bowern, Patience Epps, Jane Hill, Outi Vesakoski, Martine Robbeets, Noor Karolin Abbas, Daniel Auer, Nancy A. Bakker, Giulia Barbos, Robert D. Borges, Swintha Danielsen, Luise Dorenbusch, Ella Dorn, John Elliott, Giada Falcone, Jana Fischer, Yustinus Ghanggo Ate, Hannah Gibson, Hans-Philipp Göbel, Jemima A. Goodall, Victoria Gruner, Andrew Harvey, Rebekah Hayes, Leonard Heer, Roberto E. Herrera Miranda, Nataliia Hübler, Biu Huntington-Rainey, Jessica K. Ivani, Marilen Johns, Erika Just, Eri Kashima, Carolina Kipf, Janina V. Klingenberg, Nikita König, Aikaterina Koti, Richard G. A. Kowalik, Olga Krasnoukhova, Nora L. M. Lindvall, Mandy Lorenzen, Hannah Lutzenberger, Tânia R. A. Martins, Celia Mata German, Suzanne van der Meer, Jaime Montoya Samamé, Michael Müller, Saliha Muradoglu, Kelsey Neely, Johanna Nickel, Miina Norvik, Cheryl Akinyi Oluoch, Jesse Peacock, India O. C. Pearey, Naomi Peck, Stephanie Petit, Sören Pieper, Mariana Poblete, Daniel Prestipino, Linda Raabe, Amna Raja, Janis Reimringer, Sydney C. Rey, Julia Rizaew, Eloisa Ruppert, Kim K. Salmon, Jill Sammet, Rhiannon Schembri, Lars Schlabbach, Frederick W. P. Schmidt, Amalia Skilton, Wikaliler Daniel Smith, Hilário de Sousa, Kristin Sverredal, Daniel Valle, Javier Vera, Judith Voß, Tim Witte, Henry Wu, Stephanie Yam, Jingting Ye, Maisie Yong, Tessa Yuditha, Roberto Zariquiey, Robert Forkel, Nicholas Evans, Stephen C. Levinson, Martin Haspelmath, Simon J. Greenhill, Quentin D. Atkinson, and Russell D. Gray. 2023. Grambank reveals the importance of genealogical constraints on linguistic diversity and highlights the impact of language loss. *Science Advances*, 9(16):eadg6175.

Pia Sommerauer and Antske Fokkens. 2018. Firearms and tigers are dangerous, kitchen knives and zebras are not: Testing whether word embeddings can tell. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Brussels, Belgium. Association for Computational Linguistics.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very deep networks. In *Neural Information Processing Systems*, pages 2377–2385.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112, Montreal, CA.

John Sylak-Glassman. 2016. The composition and use of the universal morphological feature schema (unimorph schema). Technical report, Center for Language and Speech Processing, Johns Hopkins University.

Zhen Tan, Lu Cheng, Song Wang, Yuan Bo, Jundong Li, and Huan Liu. 2023. Interpreting pretrained language models via concept bottlenecks.

Hasan Tanvir, Claudia Kittask, Sandra Eiche, and Kairit Sirts. 2021. EstBERT: A pretrained language-specific BERT for Estonian.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? Probing for sentence structure in contextualized word representations. In *Proc. of ICLR*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor

Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Wolfgang Veenker. 1968. Verzeichnis der ungarische suffixe und suffixkombinationen.

Veronika Vincze, Dóra Szauter, Attila Almási, György Móra, Zoltán Alexin, and János Csirik. 2010. Hungarian dependency treebank. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

A. Virtanen, J. Kanerva, R. Ilo, J. Luoma, J. Luotolahti, T. Salakoski, F. Ginter, and S. Pyysalo. 2019. Multilingual is not enough: BERT for Finnish.

Elena Voita and Ivan Titov. 2020. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online. Association for Computational Linguistics.

Wietse de Vries, Andreas van Cranenburgh, and Malvina Nissim. 2020. What's so special about BERT's layers? a closer look at the NLP pipeline in monolingual and multilingual models. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4339–4350, Online. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Proceedings of NeurIPS 32*, pages 3261–3275. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding.

Alex Warstadt, Yu Cao, Ioana Grosu, Wei Peng, Hagen Blix, Yining Nie, Anna Alsop, Shikha Bordia, Haokun Liu, Alicia Parrish, Sheng-Fu Wang, Jason Phang, Anhad Mohananey, Phu Mon Htut, Paloma Jeretic, and Samuel R. Bowman. 2019. Investigating BERT's knowledge of language: Five analysis methods with NPIs. 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 2877–2887, Hong Kong, China. Association for Computational Linguistics.

Chuhan Wu, Fangzhao Wu, Tao Qi, Xiaohui Cui, and Yongfeng Huang. 2020. Attentive pooling with learnable norms for text representation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2961–2970.

Shijie Wu and Mark Dredze. 2019. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. 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 833–844, Hong Kong, China. Association for Computational Linguistics.

Shijie Wu and Mark Dredze. 2020. Are all languages created equal in multilingual BERT? In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 120–130, Online. Association for Computational Linguistics.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

Kelly Zhang and Samuel Bowman. 2018. Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 359–361. Association for Computational Linguistics.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 2204–2213. Association for Computational Linguistics.

Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for large language models: A survey. *ACM Trans. Intell. Syst. Technol.*, 15(2).