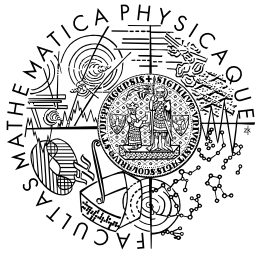

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Daniel Kondratyuk

Multilingual Learning using Syntactic Multi-Task Training

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: RNDr. Milan Straka, Ph.D.
Study programme: Informatics
Study branch: Computational Linguistics

Prague 2019

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

Prague, 10/05/2019

This thesis would have not been possible without the guidance of RNDr. Milan Straka, Ph.D., who spurred me on to experiment in his deep learning course and is what led to the creation of this thesis. I received additional astute ideas from Dr. Günter Neumann and Dr. Josef van Genabith. For that, I am very grateful for all their time and attention listening to my thought processes and giving insightful advice. I am also deeply grateful for the scholarship funding from the Erasmus Mundus program, allowing me to study abroad while also enjoying the beauties of Europe. I appreciate RNDr. Markéta Lopatková, Mrs. Bobbye Pernice, Dr. Jürgen Trouvain, and Ms. Tessa Libowski for their unwavering help in answering my endless barrages of questions. Finally, I extend my heartfelt thanks to my colleagues in the LCT, UFAL, and LST programs who made this whole experience enjoyable, my family for encouraging me, and you, dear reader, for taking your interest in looking at my thesis.

Title: Multilingual Learning using Syntactic Multi-Task Training

Author: Daniel Kondratyuk

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Recent research has shown promise in multilingual modeling, demonstrating how a single model is capable of learning tasks across several languages. However, typical recurrent neural models fail to scale beyond a small number of related languages and can be quite detrimental if multiple distant languages are grouped together for training. This thesis introduces a simple method that does not have this scaling problem, producing a single multi-task model that predicts universal part-of-speech, morphological features, lemmas, and dependency trees simultaneously for 124 Universal Dependencies treebanks across 75 languages. By leveraging the multilingual BERT model pretrained on 104 languages, we apply several modifications and fine-tune it on all available Universal Dependencies training data. The resulting model, we call UDify, can closely match or exceed state-of-the-art UPOS, UFeats, Lemmas, (and especially) UAS, and LAS scores, without requiring any recurrent or language-specific components. We evaluate UDify for multilingual learning, showing that low-resource languages benefit the most from cross-linguistic annotations. We also evaluate UDify for zero-shot learning, with results suggesting that multilingual training provides strong UD predictions even for languages that neither UDify nor BERT have ever been trained on. Finally, we provide evidence to explain why pretrained self-attention networks like BERT may excel in multilingual dependency parsing.

Keywords: syntax multilingual universal dependencies BERT

Contents

Introduction

Advances in Deep Learning have spurred the growth of several initiatives in the Natural Language Processing (NLP) community, introducing useful techniques that enable models to score higher on common metrics, predict faster, and fit into less memory. One such initiative is in multilingual learning, where a machine learning model is tasked to incorporate the information content of two or more languages to solve a task at hand. In the same way learning a new language can enhance the proficiency of a speaker's first language [Abu-Rabia and Sanitsky, 2010], a model which has access to multilingual information can begin to learn generalizations across languages that would not have been possible through monolingual data alone.

Multilingual models share a number of benefits. Works such as McDonald et al. [2011], Naseem et al. [2012], Duong et al. [2015], Ammar et al. [2016], de Lhoneux et al. [2018] consistently demonstrate how pairing the training data of similar languages (e.g., Czech and Russian, Chinese and Japanese) can boost evaluation scores of models predicting syntactic information like part-of-speech and dependency trees. Other models like Johnson et al. [2017], Ha et al. [2017] show how multilingual training enables zero-shot learning, where a model can accurately perform a task despite being given a language it has never seen before. And because a multilingual model can process multiple languages, this typically reduces the space requirements when compared to training separate monolingual models for each language.

Current NLP models have shown issues in scaling beyond a small number of languages, especially if they are distant from each other. Prior to 2019, the research community produced models trained on up to 12 languages [Johnson et al., 2017], which may be indicative of potential issues in training across more than 20 languages. This was likely due to a lack of training resources, as neural models require copious amounts of data to properly regularize them and prevent the overfitting of their input [Popel and Bojar, 2018].

But recently, the works of Howard and Ruder [2018], Peters et al. [2018], Devlin et al. [2018] have enabled several unsupervised methods that can surpass this overfitting limitation. By training model on raw text with the simple task of predicting unseen words in sentences, the models can learn contextual representations of text that can be transferred to other tasks, boosting evaluation scores. As there exists an enormous supply of unannotated raw text on the web, this can be used to regularize self-attention networks to prevent overfitting and eliminate many of the scaling issues. This forms the foundation for this thesis, which will provide a few additional key methods to scale multilingual learning of up to 75 languages on Universal Dependency parsing and show the extent to which multilingual information can boost evaluation scores over monolingual models.

The Universal Dependencies (UD) framework provides syntactic annotations consistent across a large collection of languages [Nivre, 2018, Zeman et al., 2018]. This makes it an excellent candidate for analyzing annotations across multiple languages. UD offers tokenized sentences with annotations ideal for multi-task learning, including lemmas, treebank-specific part-of-speech tags, universal part-of-speech tags, universal morphological features, and dependency edges and universal dependency labels for each sentence.

This thesis introduces and analyzes a model we call UDify, a semi-supervised

multi-task model for automatically producing UD annotations based on UDPipe Future, a winner of the CoNLL 2018 Shared Task on Multilingual Parsing from Raw Text to Universal Dependencies [Straka, 2018, Zeman et al., 2018]. By modifying a multilingual pretrained BERT model and further training it on all treebanks concatenated together, UDify is able to accurately predict UD annotations for any given language, i.e., any of 124 treebanks across 75 languages. We evaluate UDify with respect to UDPipe Future as a strong baseline, provide additional analysis for languages that multilingual training benefits prediction the most, and analyze treebanks which do not have a training set for zero-shot learning. We also use a method by Hewitt and Manning [2019] to probe the internal structure of the model for dependencies between words, showing how UDify structures its word representations very close to that of annotated dependency trees. Finally, we examine and visualize the attention heads of UDify, revealing an attention structure more sensitive to constituents like clauses and prepositions after fine-tuning.

Contributions

The main contributions this thesis provides are as follows.

1. We empirically demonstrate the first singular model, UDify, capable of predicting all Universal Dependencies annotations across all available treebanks (124 treebanks, 75 languages) while also being competitive with state-of-the-art performance.

2. We provide methods to regularize BERT fine-tuning to reduce overfitting that would otherwise prevent UDify from producing accurate predictions.

3. We observe many cases where the simple strategy of multilingual pretraining and fine-tuning benefit prediction with low-resource languages, and even languages which have never been trained on.

4. We show that fine-tuning BERT on Universal Dependencies causes its internal word representations to more closely resemble human-annotated dependency trees in both the attention heads and as a global property of its vector space.

Research Questions

This thesis will keep in mind a few important research questions related to multilingual learning for Universal Dependency parsing. Answers to these questions can help better understand how to construct massively scalable multilingual models for processing syntax.

1. What specific architectural choices allow for building a model capable of scaling to annotate Universal Dependencies across all supported languages?

2. How does multilinguality affect the performance of parsing Universal Dependencies, and which languages benefit the most from cross-lingual annotations?

3. In what ways does unsupervised pretraining help for training self-attention networks for dependency parsing?

4. What evidence is there to support why pretrained self-attention networks excel in dependency parsing?

This thesis additionally uses information from the following sources written by the author. All experiments in Chapter 3 and Chapter 4 have been conducted by the author unless explicitly indicated otherwise.

- Kondratyuk, D., Gavenčiak, T., Straka, M., and Hajič, J. *LemmaTag: Jointly Tagging and Lemmatizing for Morphologically Rich Languages with BRNNs*. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4921–4928, 2018.

- Kondratyuk, D. *75 Languages, 1 Model: Parsing Universal Dependencies Universally*. 2019.

Chapter 1 and Chapter 2 provide background information relevant to understanding UDify. Chapter 3 describes the UDify model, and Chapter 4 details and analyzes the results related to predicting UD annotations with the model. Finally, Chapter 4.5 concludes with a summary of the major results of this thesis.

1. Background and Related Work

This chapter provides a brief summary of the background information necessary to support this thesis. The background consists of an overview of the Universal Dependencies framework, neural networks, self-attention networks, contextualized word representations for transfer learning, and related work in multilingual learning.

1.1 The Importance of Processing Syntax

Syntax is the set of rules that dictates sentence structure. It is the branch of linguistics that analyzes the low-level interactions between words in a sentence [Hana, 2011]. For example, these interactions include the order of words[1],

*I wrote a thesis — *Wrote I a thesis*,

the agreement between words,

*I write a thesis — *I writes a thesis*,

the number of allowed complements,

*I handed [my thesis] [to my advisor] — *I saw [my thesis] [to my advisor]*,

and the hierarchical structure of words and phrases,

*I wrote [my thesis] [with joy] — *I wrote [my thesis [with joy]]*.

When processing any kind of natural language text, a degree of awareness of the syntax of a language is a must. The interactions of words in a language are non-trivial and can combine in many unexpected ways. When conducting a Natural Language Processing (NLP) task, the first step to better understanding the content of a sentence is most often to analyze its syntax.

Over the decades, linguists have defined formalisms, frameworks to better visualize and understand the syntax of a given sentence. These include part-of-speech, context-free grammars, constituency trees, and dependency trees. From a given example sentence one can construct these representations to see how the meaning of smaller parts of a sentence can compose to form new meaning. Rising interest in developing more sophisticated and accurate tools for processing natural language has produced models that are effective and efficient at analyzing the syntax of a given text.

Machine learning models have become increasingly popular to process syntax. A model is given a set of example sentences with annotations that it is expected to output and is tasked with learning the implicit linguistic rules to produce those annotations, even for examples the model has never seen before. To allow models to generalize well across most of the linguistic rules of a language, human annotators have produced large datasets consisting of hundreds or even millions of example sentences. One such

[1] Each sentence marked with an asterisk * is considered ungrammatical in context

collection of datasets is the Universal Dependencies corpus, which will provide the data that this thesis will use to learn models to process a series of syntactic tasks. These models will provide insight into how incorporating the syntax of not just one language but many languages can improve a model's syntactic understanding of each considered language.

1.2 Universal Dependencies

According to the Universal Dependencies Website [Zeman et al., 2018], "Universal Dependencies (UD) is a framework for cross-linguistically consistent grammatical annotation and an open community effort with over 200 contributors producing more than 100 treebanks in over 70 languages." The introduction to UD continues on,

> "Universal Dependencies (UD) is a project that is developing cross-linguistically consistent treebank annotation for many languages, with the goal of facilitating multilingual parser development, cross-lingual learning, and parsing research from a language typology perspective. The annotation scheme is based on an evolution of (universal) Stanford dependencies, Google universal part-of-speech tags, and the Interset interlingua for morphosyntactic tagsets. The general philosophy is to provide a universal inventory of categories and guidelines to facilitate consistent annotation of similar constructions across languages while allowing language-specific extensions when necessary."

The UD project grew from a need to keep annotations consistent across languages. Prior to research efforts like UD, researchers independently defined their own formal rules and tagsets to capture the syntax of their respective language. The UD annotation scheme provides an abstraction over these formats, allowing any UD-supported treebank to have the same basic rules and tags like any other treebank. This significantly reduces the complexity of models trained and evaluated on UD, as a properly constructed model can swap between any UD treebank and not need any code changes with regards to data preprocessing and data representations.

Every supported language in UD possesses one or more treebanks produced by independent research groups. Each treebank contains a list of annotated sentences formatted to suit the UD annotation guidelines. These sentences are further are split into three main files: the training, development, and test sets. The training set is used to feed directly into machine learning models to process and predict UD annotations, while the development and test sets are used purely for evaluation purposes, i.e., to test to what extent a given model is able to predict examples it has never seen before.

The training, development, and test sets are all preformatted into the CoNLL-U format. This format specifies each file to contain 10 tab-separated columns containing different morphological and syntactic annotations. These columns are represented as 6 different features of a given word. Table 1.1 provides a description and example of each feature.

The *Form* column provides the word itself, within the broader context of a sentence. The *Lemma* column provides the root form of the word form. The *XPOS* column optionally provides treebank-specific part-of-speech tags of the word forms which may be

FEATURE	DESCRIPTION	EXAMPLE
Form	Word tokens	`nominated`
Lemma	Root form	`nominate`
XPOS	Treebank-specific part-of-speech tags	`VBD`
UPOS	Universal part-of-speech tags	`VERB`
UFeats	Universal morphological features	`Mood=Ind|Tense=Past|VerbForm=Fin`
Deps	Dependency head, child, and label	`0 root`

Table 1.1: A table of Universal Dependencies features along with an example of each feature.

different across treebanks and languages, while the *UPOS* column contains language-agnostic universal part-of-speech tags. Similarly, the *UFeats* column specifies a list of universal morphological features related to the given word form. And lastly, the *Deps* columns specify dependency relations between words that are constructed in a sentence to form a dependency tree.

The following sections provide more detail into each of the UD features mentioned above and explain the broader context of how and why they are used in practice.

1.2.1 Part-Of-Speech Tags

All words have a *part-of-speech* (POS), a classification that fits into one of several structural categories. The word *fish* is a noun, *destroy* is a verb, and *incredible* is a an adjective. A linguist can classify the parts-of-speech of words in a sentence, associating one a POS tag per word. POS tags can be very coarse like *noun* or can contain fine-grained attributes of a word like *active, past participle, informal*. The Prague Dependency Treebank (PDT), for instance, employ fine-grained features in their tagset. An example can be seen in Figure 1.1 where each character in the tag represents a different feature of a word.

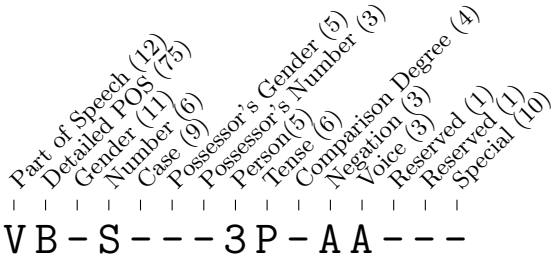

Figure 1.1: The tag components of the Czech PDT treebank with the numbers of valid values. Around 1500 different tags are in use in the PDT [Kondratyuk et al., 2018].

UD provides universal coarse-grained POS tags. Table 1.2 lists all the different types of UPOS tags that are used to classify each word across all languages.

Many treebanks in UD have been ported over from preexisting treebanks possessing their own POS tagset. As such, UD also provides an optional column for treebank-specific POS tags denoted XPOS. Figure 1.1 is an example of an XPOS tag in the Czech PDT treebank. This is the only non-universal feature provided by UD.

Part-of-speech tags are useful in that they can generalize across certain categories of words and allow one to think about a sentence more abstractly. For instance, words

OPEN CLASS WORDS	CLOSED CLASS WORDS	OTHER
ADJ	ADP	PUNCT
ADV	AUX	SYM
INTJ	CCONJ	X
NOUN	DET	
PROPN	NUM	
VERB	PART	
PRON		
SCONJ		

Table 1.2: A listing of all universal part-of-speech (UPOS) tags in the UD framework.

tagged as adjectives (ADJ) can be replaced with other adjectives without breaking the structural rules of grammar. Subjects of a sentence in English precede verbs (VERB) and are represented as nouns (NOUN). Part-of-speech tags offer an abstraction of syntax that makes it easier for linguists to study language.

1.2.2 Morphology Tags

Morphology studies how words are formed in a sentence, and how they change with respect to context. Certain inflections of a word can affect the word's meaning, e.g., in English, the ending "s" typically pluralizes a noun to indicate there are more than one of the entity the noun describes. English does not encode a high degree of morphology like many other morphologically-rich languages. Instead, English uses word order to convey much of the structural information in a sentence. But morphologically-rich languages can convey this meaning in their many word forms, and in some languages like Czech, can freely change their word order without fundamentally altering the meaning of the sentence or breaking the rules of grammar.

To capture this morphological information, UD defines an inventory of universal morphological features along nearly 50 dimensions, including case, number, gender, mood, and more. Each dimension can be assigned a value like nominative, future, first person, etc.

Morphologically rich languages are often difficult to process in many NLP tasks [Tsarfaty et al., 2010]. When processing text, one typically constructs a vocabulary of unique words that have been observed in that text. The addition of several inflectional variants across many words dramatically increases the vocabulary size of a language. The number of possible combinations of words with their morphological variants quickly explodes in size, to the point where most combinations of word forms have never been observed. This results in data sparsity and out-of-vocabulary (OOV) issues, and several additional techniques are necessary to reduce the effects of these issues. For instance, morphological tags can make it easier to determine root forms of words to reduce vocabulary sizes.

1.2.3 Lemmas

Closely related to morphology tags are lemmas. A *Lemma* of a word is its root or dictionary form. For instance, the root of "ate" is "eat." Alongside each word, UD provides a lemma column that lists corresponding lemmas of each word form. *Lemmatiza-*

tion is the process of converting a word to its root form. Lemmatization can normalize the morphology of a text, keeping vocabulary sizes manageable and making it easier to infer certain word patterns.

If one knows all the morphological information associated with a word (e.g., in a morphology tag), it should not be too difficult to come up with a rule that transforms a word into its lemma and vice-versa. However, it is not always possible, as different words can have overlapping *senses*. For instance, "found" can either be a past tense verb (*I found it*) or a present tense verb (*to found a town*). Given a word form of "found," it may be difficult to determine what sense the word form refers to. In most cases, the broader context of the sentence or text passage can clue in on what the lemma or morphological features of a word form should be.

1.2.4 Dependency Trees

Dependency parsing is the process of analyzing the lexical structure of a sentence to produce relations between words. A dependency parser works by receiving a sentence's word tokens as input and produces a dependency tree consisting of a set of directed and labeled binary relationships between words. Figure 1.2 provides an example of a dependency tree using the UD annotation scheme.

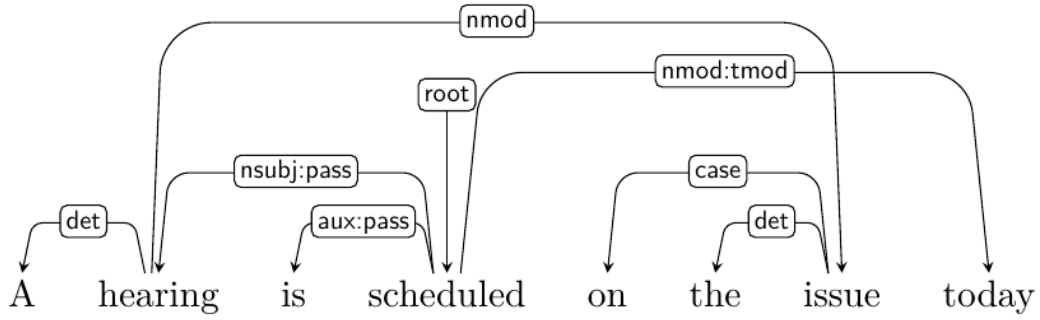

Figure 1.2: An example of a non-projective dependency tree using UD labels [Hershcovich, 2017].

A relation in a dependency tree consists of a *head* where an arrow originates and a *dependent* where the arrow terminates. Each relation arrow also carries a label that indicates the type of relationship between the head and the dependent. This can alternatively be formulated as a directed spanning tree where each word is represented as a node in the tree and each labeled edge is a binary relation (arrow).

Dependency trees also possess additional constraints to enforce a tree structure.

- There exists a root node which has no incoming edge.
- All paths from the root to any other node are unique.
- Each node except the root has exactly one incoming edge.

One special property of dependency trees is *projectivity*, where a tree can be constructed without crossing any lines. This is a nice property that makes processing language on projective sentences easier than on non-projective sentences, as projectivity allows all the dependencies to be grouped into localized compositional chunks

without any discontinuities. Non-projective dependencies can result in long-range dependencies which can be more difficult for models to predict.

UD utilizes two columns in the CoNLL-U format to provide the dependency tree of each sentence. For each word, the *head* column indicates the integer position of the head of a dependency or 0 if the word is the root of the sentence. The *deprel* column specifies the type of relation as one of several categories of labels. For instance, "amod" stands for adjectival modifier, and is a dependency relation typically used by adjectives that point to a noun. Optionally, UD also supports enhanced dependency annotations, allowing for a larger set of dependency labels.

Dependency trees reveal a higher degree of the syntactic structure of a sentence than the POS tags, morphological tags, or lemmas alone. It shows not only the function of each word in the sentence, but also the broader context of how that word plays a role in building the meaning of a sentence. Dependency trees can also be used to separate the roles of words from the order in which they are uttered in a sentence. The use of dependency trees is broadly useful for a wide variety of NLP tasks, from information retrieval and question answering [Cui et al., 2005, Lin and Pantel, 2001, Yih et al., 2013], to machine translation [Sennrich and Haddow, 2016].

The next several sections will provide the additional background necessary to understand how one may use modern methods to predict Universal Dependencies from text.

1.3 Distributional Semantics & Word Vectors

In the last several years, there has been an explosion of research in leveraging neural networks for processing natural language. The success of neural networks in processing text can be largely attributed to the *distributional hypothesis*, which in simple terms states that words in similar contexts have similar meanings. By the words of Firth [1957], "You shall know a word by the company it keeps." Using statistical modeling, one can begin to categorize the meanings and functional aspects of words with respect to their context. For instance, search engines have exploited this hypothesis by building large tables of word frequencies of documents found on the web and retrieving these documents based on similarity to user queries.

A popular use of the distributional hypothesis is to represent each word in a text with as a high-dimensional vector. Each vector contains a list of real values, each representing a feature of a vector space. This re-frames the original linguistic problem as a geometry problem, asking what series of operations are necessary to transform these word vectors into the desired result.

Figure 1.3 illustrates an example vector space of related words in 2D space. One important feature of the vector space that models exploit is a *similarity measure*. For instance, this can be is cosine similarity, where vectors that point in a similar direction are said to be semantically similar, i.e., convey nearly the same meaning. This gives rise to simple and intuitive linear vector operations on words that preserve semantic relatedness between them, e.g., $king - man + woman = queen$.

Words can be represented as vectors in any number of ways, including word frequency and co-occurrence, or using more advanced deep learning methods to produce *word embeddings*. Modern word embeddings methods were popularized by Mikolov et al. [2013a], which provided a method to produce word vectors by predicting the

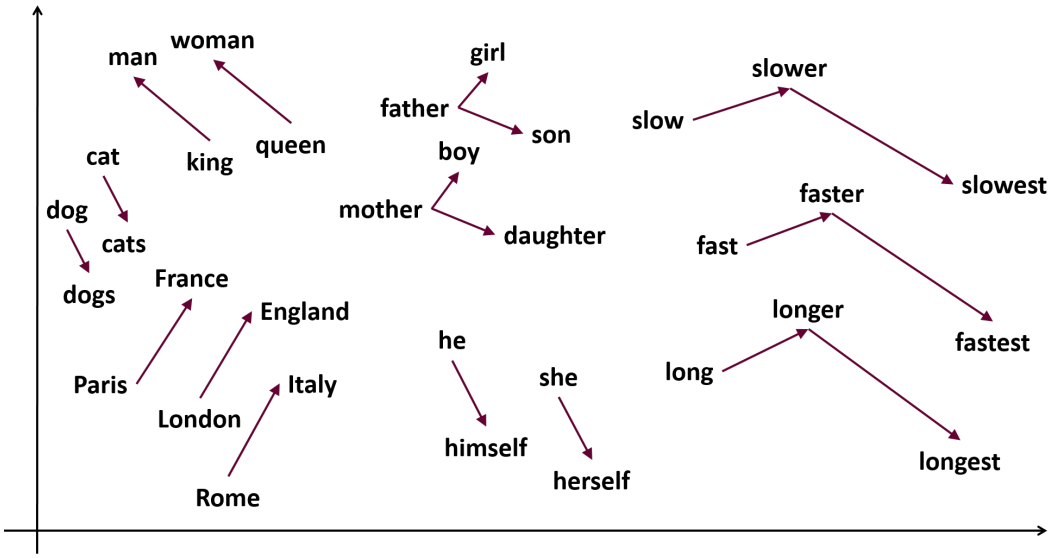

Figure 1.3: An example of a word-based vector space with semantically related words pointing in the same direction [Zafranyj, 2019].

context in which each word appears. By iteratively refining these estimates, a high-quality vector representation of a vocabulary of words can be produced and used in many NLP models. Most models which use neural networks to process text also use word embeddings of some kind, particularized to the task that they are used in.

1.4 Neural Networks

This thesis will focus on utilizing special representations of word vectors that demonstrate superior performance across a variety of metrics, and adapt them for parsing Universal Dependencies. The next sections will provide further background into how to calculate these special representations and furthermore how to use them for Universal Dependency parsing.

1.4.1 Deep Learning

Deep Learning has grown in popularity in the greater NLP community due to its flexibility, scalability, and distinguished performance across an increasingly wide variety of NLP tasks [LeCun et al., 2015]. It provides a framework for solving certain kinds of problems by training *universal function approximators* to solve a given task. By providing these approximators a large number of examples of inputs and outputs, they can begin to correlate certain relationships in the input to produce the correct output. This forms the basis of machine learning, and deep learning provides an additional set of tools, i.e., neural networks, to create and iteratively improve such function approximators.

Given an input, a neural network applies a series of weighted operations on that input to produce an output. These weights, also called *parameters*, are not tuned explicitly, but are updated automatically through implicit relationships in the provided input data. By feeding the network more examples, it can refine its parameters so that it approaches closer to generalizing across all the given examples. Initially, a neural

network is given a (relatively) random arrangement of parameter weights which are then continually updated until a stopping criterion is met. Typically, examples are batched together, computed in parallel, and then averaged. The idea is to provide a better statistical representation of the entire dataset than using just one example which can be very noisy.

An *optimizer* is used in conjunction with the neural network, which specifies how those parameters should be updated with every successive iteration of example data. The optimizer is paired with a *loss function*, which dictates the amount of error the network produces with respect to each input example. The goal of the optimizer is to minimize this error or loss. However, this is no guarantee that the network will learn to find patterns and generalize across the data and instead may overfit to the training set. Sufficient regularization is required to reduce overfitting, and the later sections will detail some of the important regularization techniques necessary for good generalization performance.

1.4.2 Feedforward Neural Networks

Several neural architectures have been developed to handle various kinds of input data and exploit their properties to more accurately solve the given problem. The feedforward neural network, otherwise known as the multilayer perceptron (MLP), is one of the simplest and most commonly used neural architectures.

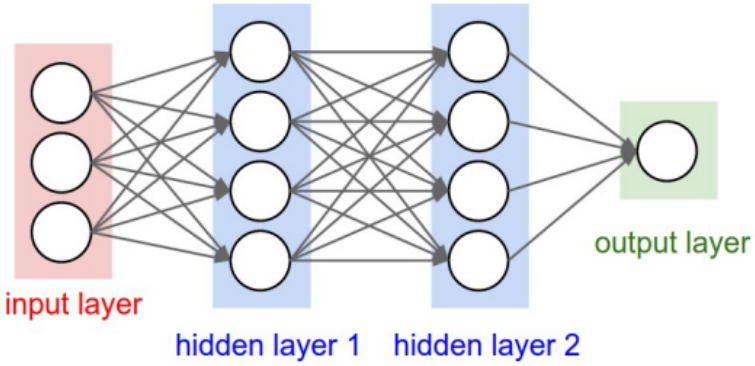

Figure 1.4: An example of a feedforward neural network with two hidden layers [Karpathy, 2019].

Figure 1.4 illustrates an example feedforward neural network with an input layer, an output layer, and two hidden layers. A layer or state is said to be *hidden* if it is internal to the network itself, whose parameters do not directly relate to the inputs or outputs. Each layer consists of a number (or dimension) of neurons that are fully connected to all the neurons of the previous layer. Each neuron receives a weighted sum of all outputs of neurons it is connected to in the previous layer plus some bias factor, producing a new scalar output for that neuron. But before sending this output to the next neuron, networks typically introduce a non-linear *activation function* that distorts the output. Instead of summing each neuron individually, the mathematical operations can be expressed as matrix operations. For example, suppose that $\boldsymbol{x}$ is the vector corresponding to the input layer outputs, $\boldsymbol{W}$ is a matrix of weights providing a weighted sum for each input, and $\boldsymbol{b}$ is a simple bias term after the weighted sum. Then

the output of all neurons in hidden layer 1 $\boldsymbol{h}$ can be written as the vector

$$\boldsymbol{h} = ReLU(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1.1}$$

where and where $ReLU$ is a popular non-linear activation function defined as

$$ReLU(x) = max(0, x) \tag{1.2}$$

By applying successive hidden layers in this fashion, a properly trained network will produce a representation of hidden parameters such that for any input, it can produce a reasonable guess at the output. Initially, these weights are randomized and require successive weight updates to minimize the loss function that identifies the network's error.

Feedforward neural networks have a notable deficiency that makes it hard to process text on their own: they do not possess any inherent mechanism to capture the *order* of input data. One can attempt to apply word vectors as an input to a feed-forward network to perform a given task on text, but notice that the implicit ordering information in the input layer is lost after summing up the values of the input layer to produce the hidden layer. In effect, this would produce a bag-of-words model that is only sensitive to the types and frequencies of the input word embeddings.

1.4.3 Recurrent Neural Networks

To allow a neural network to explicitly model word order, one popular neural architecture that has been devised is the recurrent neural network (RNN). As opposed to a feedforward network, RNNs persist state for multiple durations of their input. Given an input, an RNN computes an intermediate state and uses this information when computing the next state for the next input, in turn affecting the output state. One can view an RNN as a kind of memory cell that can remember and forget pieces of information as it receives new input. RNNs are the most widely used architectural component when processing words in a text, as they lend themselves well to sequential data. Word vectors can be input to RNNs sequentially, producing a sequence of output vectors, one output vector produced per input, while the hidden states related to previous inputs and states are passed along.

One widely-used type of RNN cell is the Long Short Term Memory (LSTM) unit seen in Figure 1.5 [Hochreiter and Schmidhuber, 1997]. It provides a series of operations on the input that control information flow. Another widely-used type of RNN is the Gated Recurrent Unit (GRU), which provides a slightly different formulation to calculate its hidden and output states [Chung et al., 2014]. We only briefly mention them here and do not detail the specific operations as seen in the figure, as they will not be necessary to discuss the main model introduced in the core of this thesis. The important point is that the operations performed in an RNN are purposefully built to help a neural network remember past inputs and use this information to better process output states for the current input.

1.4.4 Self-Attention Networks & the Transformer

Prior to 2017, state-of-the-art models in NLP were dominated by recurrent neural networks. Machine translation systems used a popular architecture known as *sequence-to-sequence*, which takes an input sentence word-by-word through an RNN encoder,

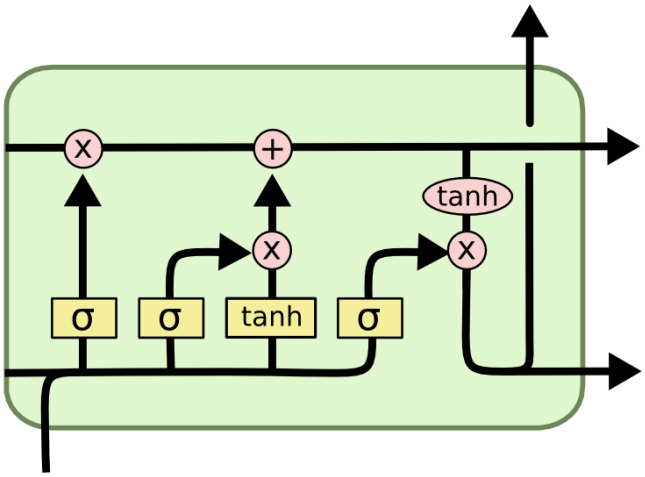

Figure 1.5: A visual representation of the operations performed inside an LSTM unit [Olah, 2015].

produces a single vector representing the information content of a sentence, and uses another RNN decoder to output a sentence translation word-by-word.

Then Vaswani et al. [2017] released a new architecture for machine translation called the Transformer or the self-attention network. The authors completely eliminate RNNs as an architectural feature, and introduce several architectural concepts like multi-head self-attention and positional encoding to create a novel architecture that achieved (and continues to achieve) state-of-the-art performance in machine translation.

Scaled Dot-Product Self-Attention

According to Vaswani et al. [2017], self-attention is an attention mechanism relating different positions of a sequence to produce a representation of each part of the sequence. The use of self-attention has been observed in prior works [Cheng et al., 2016, Parikh et al., 2016, Paulus et al., 2017], but the work of Vaswani et al. [2017] was the first to focus solely on self-attention as a method to create vector representations for sequence processing, without requiring any recurrent or convolutional components.

The self-attention mechanism (see Figure 1.6) consists of three primary inputs: the *query* $\boldsymbol{Q}$, the *key* $\boldsymbol{K}$, and the *value* $\boldsymbol{V}$. The scaled dot-product attention $\mathrm{Attention}$ computes the dot products of the query with all the keys, scales them, and applies a softmax function to obtain the weights on the values. In practice, the queries, keys, and values are bundled together into matrices, as this operation will be used to compute dot-products of a query with respect to all keys and further across all possible queries.

To put it more formally,

$$\mathrm{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V} \tag{1.3}$$

where d_k is the dimension (number) of the keys. The $softmax$ function is defined

as

$$\text{softmax}(\boldsymbol{\alpha}_i) = \frac{e^{\boldsymbol{\alpha}_i}}{\sum_j e^{\boldsymbol{\alpha}_j}} \tag{1.4}$$

where e is the natural constant. In this instance, the softmax is a way to normalize a vector of values with respect to log-space.

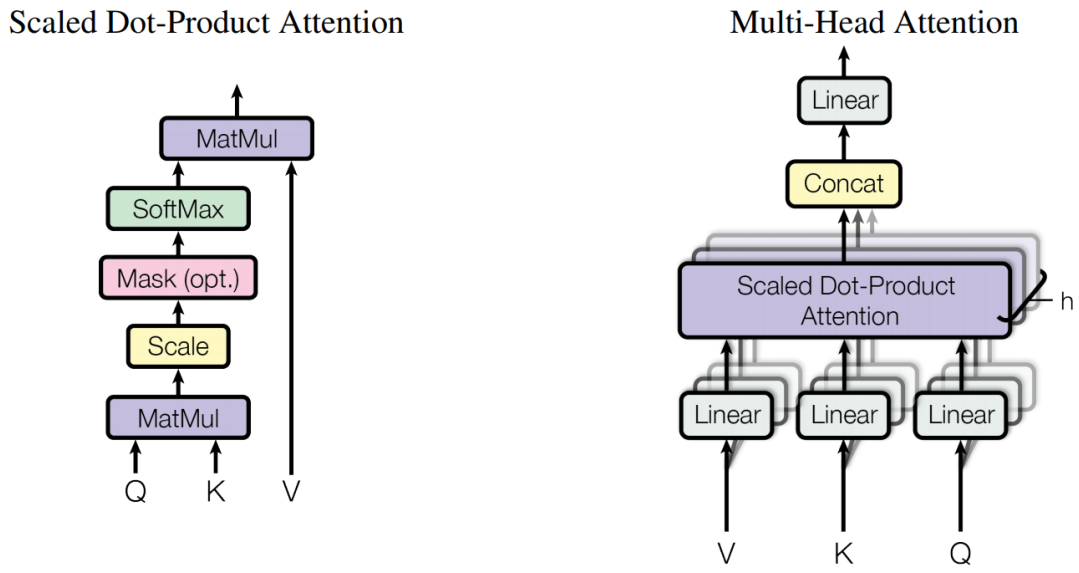

Figure 1.6: Transformer self-attention head (left) consists of a *query* $\boldsymbol{Q}$, *key* $\boldsymbol{K}$, and *value* $\boldsymbol{V}$. Multi-Head Attention (right) combines h of these heads together, running them in parallel [Vaswani et al., 2017].

Intuitively, one can think of the scaled dot-product attention as a kind of continuous lookup table for vectors. Given a query vector and key vector, they are multiplied together in a dot-product, which means that the output will be a large scalar if the vectors match very closely to each other (small angle between the vectors), equal to 0 if the vectors are orthogonal, and negative if the vectors are very dissimilar (an angle of more than 90 degrees). By considering a sequence of keys and multiplying them with the query, this produces a set of scalars which indicate the "strength" to which the key matches with the query. These scalars (after softmax normalization) are then multiplied by their respective value vectors to scale each value based on how closely the key matched the query. In other words, high key-query similarity will produce a high corresponding value that will be passed along as useful information, while low similarity will be less likely to make a contribution to the final output.

This dot-product attention is computed between every pair of words in a sentence, where the queries, keys, and values all represent the same sequence of (projected) vectors. See Figure 1.7 for a visualization of the attention mechanism with respect to an example sentence. The words on the left represent the keys of the sentence, and the words on the right represent the keys. Before the calculation, the word vectors are first projected to different vector spaces. For a particular word representing a query, this is multiplied with all the keys of the sentence, and the highest similarity will produce the highest weight.

Instead of computing a single attention function, one can compute multi-head attention, where h separate self-attention operations are computed in parallel and whose

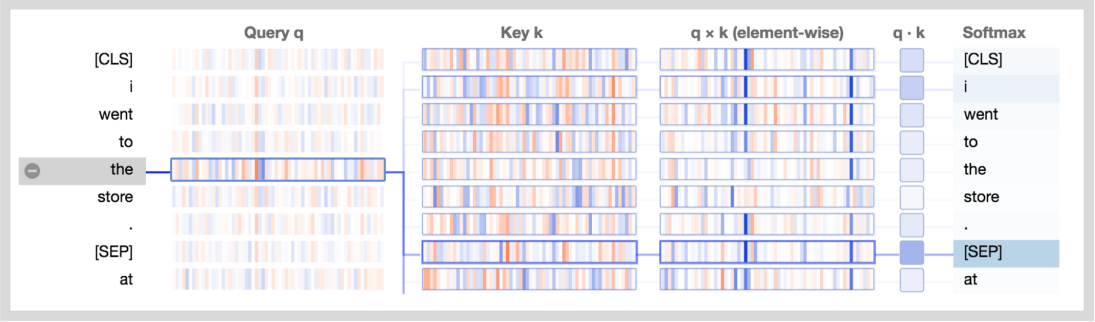

Figure 1.7: A visualization of the self-attention mechanism working on an example sentence [Vig, 2019]. High positive and negative values are illustrated as blue and orange bars respectively.

outputs are concatenated together. Note that before the dot-product operation, all inputs are first projected through separate linear (feedforward) layers so that the input vectors can be "viewed" differently with respect to each attention head. Otherwise, the highest dot-product would always be the query dot-product with respect to itself, as they are the same vector and therefore have the highest similarity.

Encoder Layers

The encoder of the Transformer model consists of a number of encoder layers stacked on top of each other as seen in Figure 1.8. Each layer has two sub-layers, the multi-head self-attention mechanism, and a simple position-wise feed-forward network with ReLU activation. The layer employs residual connections, adding the input prior to computing the sub-layer to the output after the sublayer. All the outputs are then normed with layer normalization [Lei Ba et al., 2016]. The final output of each layer is a series of vectors, one for each position of each input word.

The Transformer model also possesses a separate decoder with masked multi-head attention, but this thesis will only focus on the encoder architecture as thus described.

Positional Encoding

Another relevant aspect of self-attention networks is the positional encoding scheme. The self-attention encoder has no recurrence and therefore by itself has no inherent information related to the order of the inputs. To inject some information about the positions of each word, the authors add a positional term to each input embedding that is a function of the position of each word.

Figure 1.9 plots the different values of the positional encoding of the first 20 word vectors in a sentence assuming each word vector is of dimension 512. The authors use sinusoidal functions to calculate the positional embedding and sum it with the word vector. The authors chose this particular formulation, as each positional vector with an offset can be represented as a linear combination of the positional vector with the position of the offset vector, and so the network can learn relative positions between words.

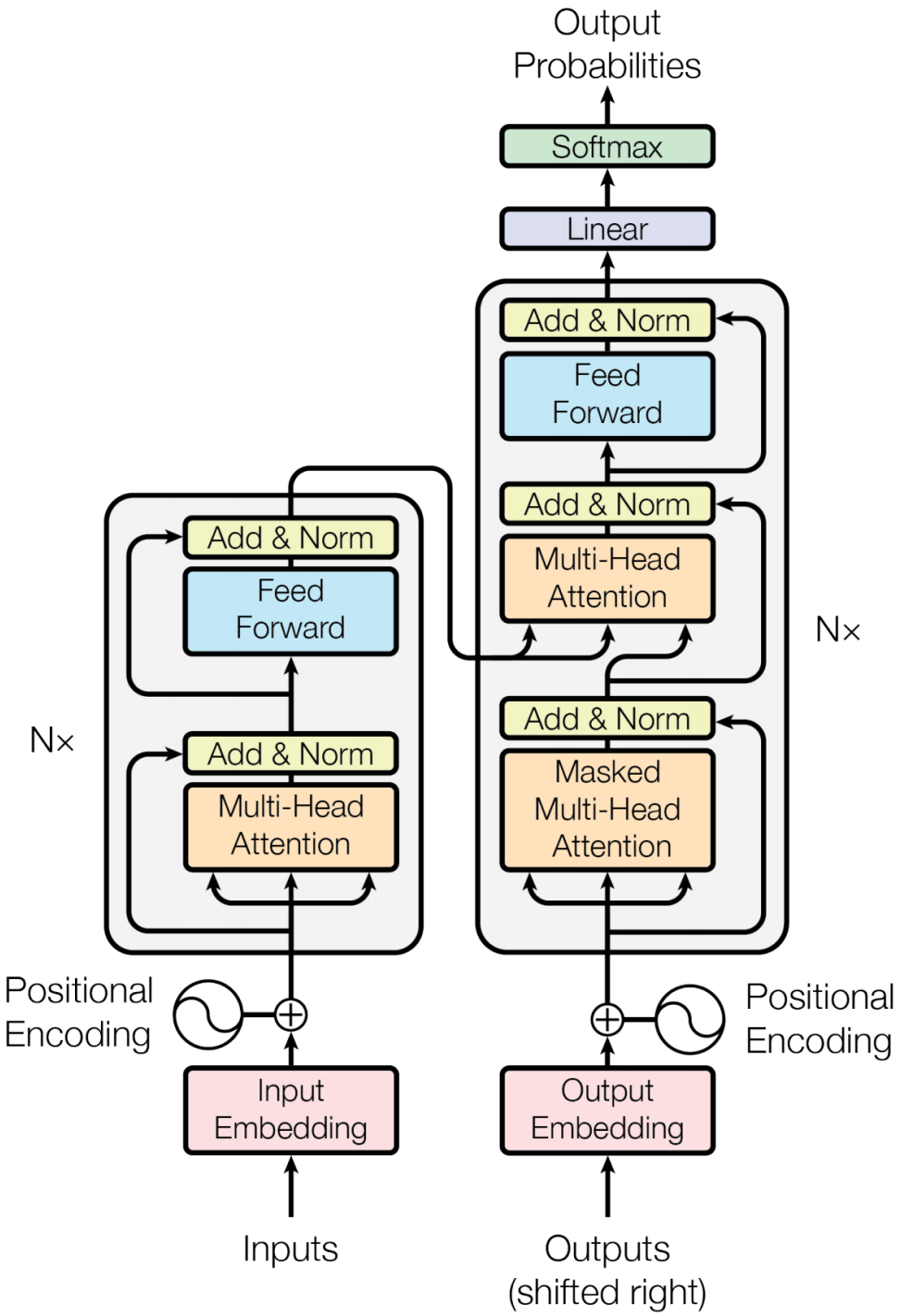

Figure 1.8: Transformer self-attention model introduced by Vaswani et al. [2017] used for machine translation.

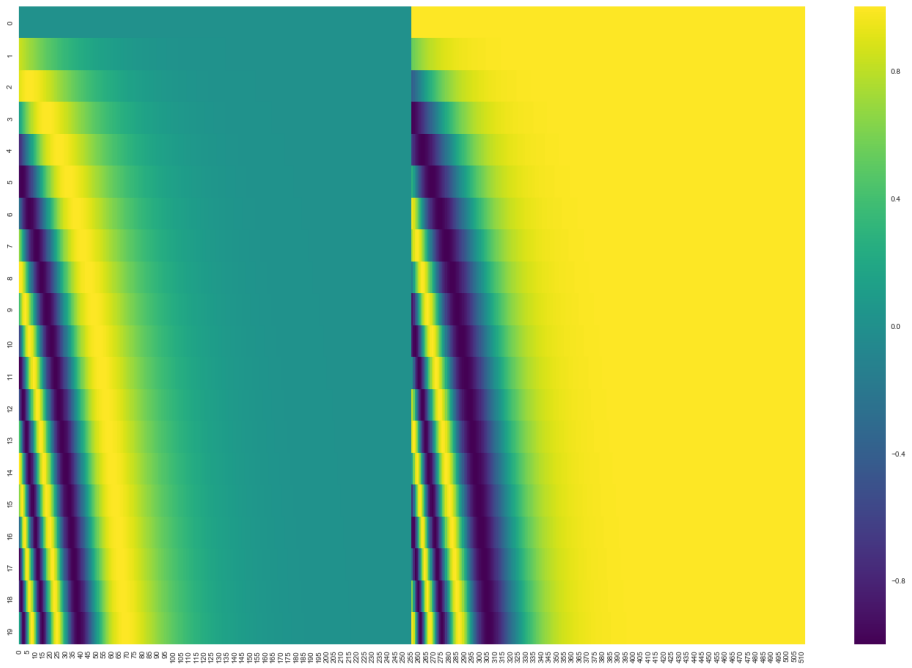

Figure 1.9: Positional encoding for 20 words (y-axis increasing downward) with an embedding dimension of 512 (x-axis increasing rightward). Lighter colors indicate activations closer to 1, and darker colors indicate activations closer to -1 [Alammar, 2018].

Byte-Pair Encoding (BPE) & Wordpieces

Up until this point, all neural processing has been dealing with operating on word vectors. However, a major disadvantage of using word vectors is dealing with large vocabulary sizes and new unknown words not seen in the vocabulary. The larger a dataset becomes, the more unique vocabulary items will be observed, and this will continue unbounded. This is a natural consequence of Ziph's law [Zipf, 1949, Fagan and Gençay, 2011], and creates problems in storing observed words and handling previously unobserved words.

To alleviate some of the effects of this problem, researchers have developed and used methods for generating *subword units*. As more text is processed, it becomes increasingly likely that new words are simply inflected variants of previously known words. By breaking apart words into their smaller constituent parts, models can mix and match different subword units to represent new words without having to increase their vocabulary, so long as those individual units have been observed before. This is especially helpful for morphologically-rich languages, which contain many inflected versions of the same underlying root word.

One popular method to generate such subword units is to use the *byte-pair encoding (BPE)* scheme. Introduced by Gage [1994] and adapted by Sennrich et al. [2015] for use in machine translation, the idea is to start from characters and iteratively combine them based on word frequency to create larger and larger units until a stopping criterion is met. Initially, all subword units are just represented as characters. Then through each successive iteration, the most commonly consecutively occurring units are concatenated together, forming a new unit in the vocabulary. The algorithm continues until a sufficiently large vocabulary of units has been reached. Once completed, text can be tokenized to fit these subword units. Wu et al. [2016] introduce a sim-

ple language-independent tokenizer to produce such BPE-generated units given raw text, and apply some additional conventions in processing to handle word boundaries like whitespace. Variants of this tokenization scheme for producing subword units are commonly paired with self-attention networks [Kudo and Richardson, 2018], and the output of such a tokenization scheme to produce BPE-defined units are commonly called *wordpieces*. For instance, `definitely` can be split into units `definite` and `##ly`. Vaswani et al. [2017] use the convention that all subword units that are continuations of a previous unit within a word are prefixed with the characters `##` to make it easier for a model to identify word boundaries (e.g., `ly` would be considered the start of a new word).

One could think of just representing all text with just characters to bypass this problem entirely. However, this approach is usually empirically worse for evaluation performance, as not only do neural networks need to identify word boundaries, but this also greatly lengthens the maximum number of dependencies between words from less than a dozen steps to potentially hundreds. There has been work in improving purely character-based models, e.g., for machine translation [Lee et al., 2017], but current models have yet to demonstrate performance on par with the state-of-the-art. In practice, a combination of subword units and character-level recurrent units can be used for the greatest boost in performance.

Inverse Square-Root Learning Rate Decay with Linear Warmup

When training any neural network, one must be conscious of the *learning rate*. This hyperparameter is what scales the gradient update (i.e., the degree to which the weights are updated) globally. By increasing the learning rate, the gradient updates increase proportionally, causing a greater change in the weights. The effect of learning rates on model performance has been widely explored [Smith and Topin, 2018, Vaswani et al., 2017, Howard and Ruder, 2018], showing that decaying (i.e., slowly reducing) the learning rate benefits a network's evaluation performance.

In the Transformer paper, Vaswani et al. [2017] propose the use of an *inverse square-root learning rate decay* scheme (otherwise referred to as Noam learning rate), where the learning rate is first rapidly increased for a number of training iterations and then gradually decreased for the remainder of training. As opposed to recurrent networks, self-attention models introduce training instability that can make it difficult for the model to learn a good set of initial parameters early on in training. To combat this, the model is first trained with a linear warmup for a few hundred iterations and then decreases proportionally to the inverse square root of the current number of iterations. The idea is to allow the model which starts from a bad set of random parameters to slowly adjust to better parameters before making larger parameter updates. More specifically, this learning rate scheme is defined as

$$learning\_rate = d_{\text{model}}^{-0.5}\min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (1.5)$$

where $step\_num$ is the current iteration number, $warmup\_steps$ is a hyperparameter specifying the number of steps to warm up before decaying, and d_{model} is a hyperparameter scaling the learning rate proportional to the size of the model.

A visualization of such a learning rate scheme is given in Figure 1.10.

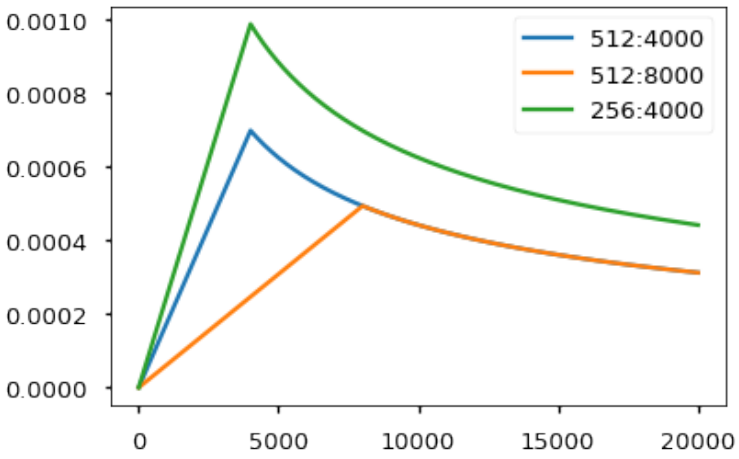

Figure 1.10: A plot of the inverse square-root learning rate decay scheme applied to Transformer models with different hyperparameters, where the x-axis represents the number of training steps and the y-axis represents the learning rate [NLP, 2018]. With respect to the different plots, the left number represents a global factor d_{model} scaling the learning rate, and the right number represents the number of steps $warmup\_steps$ to warm up before decaying.

Self-Attention Networks for Processing Syntax

One natural observation arising from the introduction of the Transformer model is that self-attention could be used for tasks other than machine translation. But in the years since the introduction of the state-of-the-art transformer model which continues to provide superior machine translation score over recurrent networks, few models have shown the same for applying the same basic self-attention architecture for other tasks. For instance, Strubell et al. [2018] define a self-attention multi-task network for semantic role labeling by predicting intermediate syntactic tasks. In lower layers, the network predicts a dependency tree over the input sentence and uses this tree to guide the network attention heads to attend to dependents for each given word. While this does show to improve performance for the downstream semantic role labeler, one notable result is that the self-attention parser produces lower quality dependency trees than with recurrent (LSTM) networks, and so results in slightly lower semantic role labeling performance.

A major difficulty in adapting self-attention for NLP tasks other than machine translation lies with adequate regularization. Self-attention networks tend to overfit much more than their RNN counterparts, meaning that stronger regularization is necessary. Machine translation does not have this problem to the same degree, as enormous datasets of translated sentences are usually available for the most widely used languages. By seeing more training examples, the network can rule out certain patterns in the data that do not hold in general and better represent the task at hand. However, this regularization does not apply to most syntactic NLP tasks like parsing Universal Dependencies, as most annotated datasets are comparatively small. Therefore, alternative regularization techniques are required. The next section will provide a notable technique in contextualized word representations which will allow the use of external resources to regularize self-attention networks.

1.5 Transfer Learning with Contextualized Word Representations

The landscape for developing powerful new neural models for NLP is quickly evolving from isolated components trained on single datasets to models that learn from multiple sources of information and can transfer this knowledge to new domains. In 2018, the release of several landmark papers led to the development of scalable, multi-purpose contextualized word embeddings that can quickly transfer their knowledge about raw text to any other NLP model that also utilizes word embeddings [Howard and Ruder, 2018, Peters et al., 2018, Devlin et al., 2018].

The presence of raw text resources on the web completely dwarf the total collection of available annotated datasets. It would be especially useful if the raw text could be exploited using unsupervised methods to improve existing models. More specifically, neural models could benefit from using embeddings that somehow incorporate the types of information seen in arbitrary text. This would greatly widen the available resources for training word embeddings from a potentially tiny dataset to a massive collection of linguistic data. By pretraining word embeddings using the statistical properties of the words in an arbitrary text, the word embeddings can learn properties of the language and observe rare vocabulary items that otherwise would not have appeared in a given dataset, and this information can then be transferred to existing models that use word embeddings. Clark et al. [2018] report that semi-supervised learning methods scale well enough that they are capable of enabling the development of larger and more sophisticated models for NLP tasks with limited amounts of labeled training data. The following sections will provide a brief history of recent methods in pretraining word representations for knowledge transfer to provide insights into how these methods can regularize self-attention networks.

1.5.1 Word2Vec: Unsupervised Pretraining of Word Embeddings

Mikolov et al. [2013a,b] introduce methods, called *word2vec*, to pretrain word embeddings on unsupervised text using statistical properties of that text. Motivated by the distributional hypothesis, the skipgram method trains word vectors by using a simple predictor to predict the surrounding context of a word, i.e., the nearby words that appear alongside the word in the text. This simple unsupervised method has been shown to create word embeddings that can transfer information to other neural models that use word vector representations as inputs, boosting performance.

While word2vec does capture global contextual information with respect to each word, one important observation is that when these word vectors are represented as a sequence of words in a sentence, the word vectors are independent of each other and carry no direct contextual information with respect to that sentence. A word2vec embedding vector provides the exact same information no matter what sentence it is in. It provides no dependency information with respect to any other words in the sentence, which can erase a lot of information with respect to the unsupervised text resources the embedding was trained on. The following sections detail more recent methods that incorporate contextual dependencies between word vectors to improve knowledge transfer over word2vec.

1.5.2 ELMo: Deep Contextualized Word Representations

The ELMo model (Embeddings from Language Models) [Peters et al., 2018] introduces the notion of deep contextualized word representations, where word embeddings are calculated not only based on the surrounding context of other words, but more explicitly by the sequence of words that directly surround the word in a sentence. The authors apply a simple method used in neural *language models*: given a sequence of words, predict the next word. This is easily calculated in recurrent neural networks as a prediction of the next word from an output state pertaining to the inputs of all previous words up to the current word.

The ELMo model trains two language models in parallel, a forward language model by processing words left-to-right as usual in an RNN, and a backward language model predicting words right-to-left by reversing the order of words. The model then stacks multiple recurrent layers on top of each other much like a multi-layer feedforward network in hopes that the network can compute hierarchical representations of text. Finally, the model computes a weighted sum along each layer, allowing the network to mix together different hidden states for a given word to better incorporate different views of the context.

By training ELMo on sentences of raw text, the model learns different embedding representations of each word based on the context of the surrounding words in the sentence. The authors show that using these embeddings empirically result in better evaluation performance when transferred to many neural NLP models than using non-contextualized representations like word2vec.

1.5.3 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Shortly after the release of ELMo, Devlin et al. [2018] introduced BERT, a model providing contextualized word representations that broke several records in many NLP tasks, and continues to provide state-of-the-art performance. BERT can be thought of as a natural extension of ELMo for self-attention networks with a few additional tweaks. BERT is essentially a slightly modified version of the self-attention encoder of the Transformer network applied to language modeling.

Note that BERT does not calculate a language model in the traditional sense, but instead calculates a *masked language model*. While a standard language model is used to calculate the probability of the next word given the previous context of the text, a masked language model utilizes both directions and therefore cannot be directly used as a standard language model due to being able to "peek" into the future and cheat its predictions.

Instead, BERT takes the simple approach of masking a small proportion of input words and train the network to predict those masked words. The model masks words by randomly selecting wordpieces with 15% probability and replacing them with a special `[MASK]` token. The goal of training BERT is to get the self-attention network to learn the contexts of words with respect to their surrounding words in a sentence. This allows the network to learn not only semantic relatedness of words as in word2vec, but also the implicit syntactic rules necessary to build a grammatically likely sentence. As will be seen in the coming chapters, this simple training procedure encodes a very rich representation of words that makes it easy for NLP models to fine-tune these

representations to suit their purposes.

In addition to inputting a sequence of tokens representing a sentence, BERT also adds a *next sentence prediction* task, where the network is given two sentences separated by a `[SEP]` token and tasked with predicting whether the second sentence directly follows the first sentence in a text. To aid this process, the model adds a special `[CLS]` token to the beginning of each input sentence and applies a binary classifier to the final layer at that token position for next sentence prediction. To differentiate between sentences, the network also trains two segment embeddings that are summed with each respective sentence. An example of the final input representation can be seen in Figure 1.11. This allows BERT to accumulate information in each successive layer to better capture not only dependencies between words in a sentence, but also words between sentences. The authors show how the resulting embedding corresponding to the `[CLS]` token acts as a sentence embedding representing the content of the entire sentence.

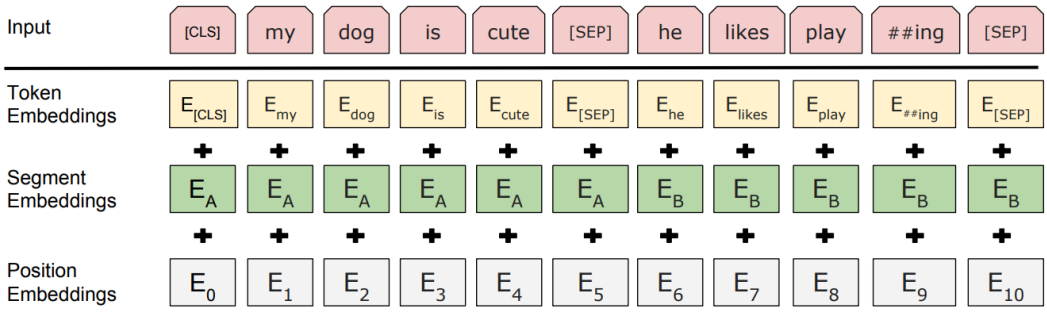

Figure 1.11: BERT input representation consisting of a sum of wordpiece token embeddings with special separator tokens, segment embeddings for sentence A or sentence B, and relative positional embeddings [Devlin et al., 2018].

Devlin et al. [2018] extract article text from the entirety of English Wikipedia and use it as the unsupervised training data for BERT. They released two sizes of the model: a base model with 12 layers, 12 attention heads per layer, and hidden vector dimensions between each layer of 768, and a large model with 24 layers, 16 attention heads per layer, and hidden dimensions of 1024. And by applying the training scheme described above, the authors demonstrate an empirically powerful model that generates contextualized embeddings that can be applied to many NLP tasks to achieve state-of-the-art performance.

To gain the best performance boost, the authors suggest fine-tuning BERT to each task. Figure 1.12 illustrates an example of fine-tuning BERT for use in named entity recognition. Upon generating contextual embeddings with BERT, a feedforward classifier can be trained and applied to these embeddings to predict named entity tags with respect to each word. But in addition to training the classifier, the parameters of BERT can be fine-tuned as well, updated with respect to the gradients of the loss calculated in the classifiers. While BERT was originally trained with a base learning rate of $1e^{-4}$, better training stability can be achieved when using a lower learning rate like $3e^{-5}$ for fine-tuning. By fine-tuning in this manner for a few epochs, BERT will alter its contextualized representations to best fit the given task, demonstrating high scores for named entity recognition, sentiment classification, question answering, natural language inference, and more [Devlin et al., 2018]. It can be readily apparent that the high volume of text resources used to train BERT are what allow the self-attention network to be

successful in not overfitting to its input data. This simple pretraining strategy provides an excellent starting point for fine-tuning to new tasks like dependency parsing, as the attention heads are properly regularized to recognize important dependencies between words.

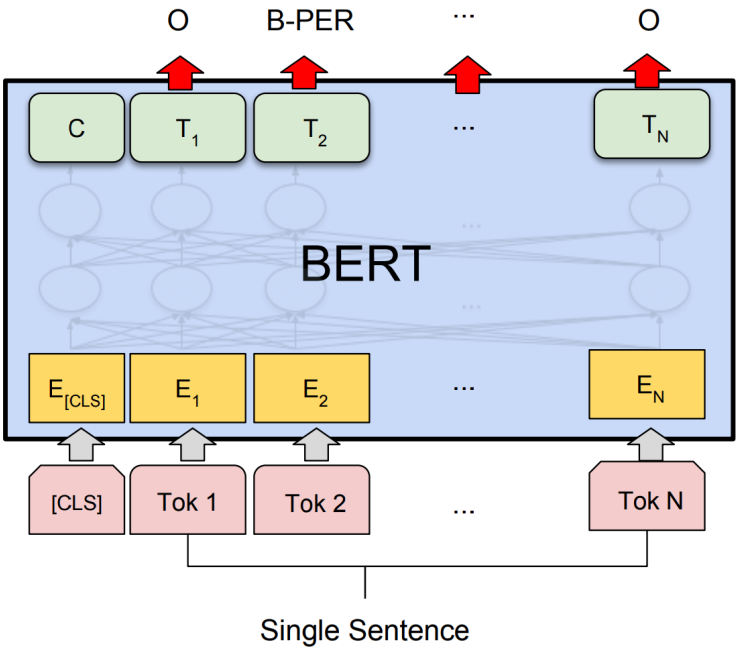

Figure 1.12: BERT model adapted for sequence tagging tasks [Devlin et al., 2018].

1.5.4 ULMFiT: Universal Language Model Fine-tuning for Text Classification

Howard and Ruder [2018] introduce several key techniques for fine-tuning contextualized word representations, which this thesis will exploit to achieve better evaluation performance. The authors introduce several relevant concepts:

- **Discriminative fine-tuning**, the process of applying different learning rates to different layers. As different layers process different kinds of information, they should be fine-tuned to different degrees. Higher layers capture less general information than lower layers [Yosinski et al., 2014], and so can change faster without introducing instability.

- **Gradual unfreezing**, where all layers of the network are frozen (held constant) except the final layer, and then iteratively unfrozen after each epoch. The main idea is to allow the network to gradually let the top layers converge first before the more general lower layers that they depend on, as updating low layers too quickly can risk catastrophic forgetting of the pretrained task.

- **Slanted triangular learning rate**, where the learning rate quickly increases linearly and then decays linearly at a lower rate, as a function of the number of training iterations. The goal is to get task-specific features to quickly converge to reasonable parameters early on in training, similar to the inverse square-root decay introduced earlier.

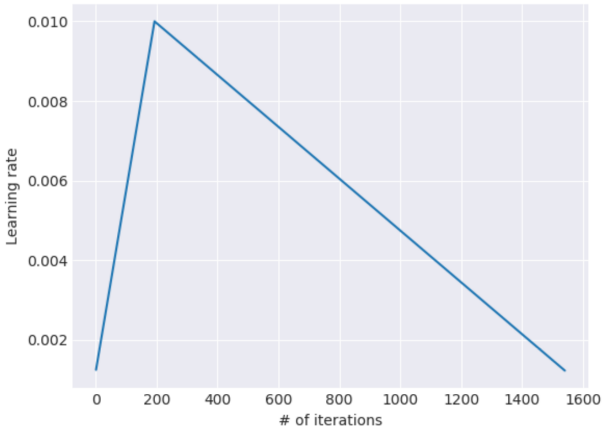

Figure 1.13: A visualization of the slanted triangular learning rate proposed by ULMFiT [Howard and Ruder, 2018].

Combined together, these methods allow for faster convergence and higher evaluation performance when using language models for fine-tuning. These methods will become important when this thesis investigates strategies for fine-tuning BERT on Universal Dependencies.

1.6 Multilingual Learning

Up until this point, this chapter has discussed purely monolingual models. Models of this type are exclusively trained and evaluated on a language of choice and do not incorporate any information related to other languages. For English, this does not tend to pose a problem, as resources for a given NLP task tends to be abundant in English compared to many other languages. However, the vast majority of languages are low-resource, making it difficult for neural networks to learn adequate generalizations to perform tasks in these languages.

Multilingual learning presents an attractive compromise to collecting larger datasets for low-resource languages. A multilingual model may be able to learn from similar languages to improve its performance in a target language, reducing the overfitting difficulties introduced with low-resource data. In addition, a multilingual model trained on many languages can further reduce the amount of space required to save all models to disk.

Recent work has indicated that sharing training sets of similar languages for multilingual parameter sharing can improve not only syntactic tasks like dependency parsing [Naseem et al., 2012, Duong et al., 2015, Ammar et al., 2016, de Lhoneux et al., 2018], but also more complex tasks like neural machine translation [Dong et al., 2015, Firat et al., 2016, Lu et al., 2018]. Combining Universal Dependencies treebanks in a language-agnostic way was first introduced in Vilares et al. [2016], which train bilingual parsers on pairs of UD treebanks, showing similar improvements. More recently, Mulcaire et al. [2019] have shown a method to produce multilingual contextual word representations by training a single language model on text from multiple languages, and use them in dependency parsing. The authors leverage ELMo pretraining, then using produced embeddings to train bilingual models.

Combining all the prerequisite knowledge above, this thesis will use these concepts to define a multilingual neural model capable of parsing Universal Dependencies for

any language.

2. Predicting Universal Dependencies from Raw Text

The following chapter introduces the task of parsing Universal Dependencies in the context of the CoNLL 2018 Shared Task. It identifies several notable methods for producing accurate parsing models, highlights important prior models, and explores current work in multilingual modeling for Universal Dependency parsing. This will form the basis for a new model called UDify which will show improvement in several aspects over these previous models in Chapter 3.

2.1 The CoNLL 2018 Shared Task

The CoNLL 2018 Shared Task in Multilingual Parsing from Raw Text to Universal Dependencies provided a competition for participating teams to generate all UD annotations (UPOS, XPOS, UFeats, Lemmas, and Deps) from raw text. The given text input was assumed to be non-segmented and non-tokenized, and the participants were required to predict their own segmentation of words and sentences before predicting UD annotations. The sections below will highlight notable ways of predicting each UD task that is relevant to the UDify model introduced in the next chapter.

2.2 Part-of-Speech Tagging

Given an input sequence $X = x_1, x_2, \ldots, x_n$, the goal of sequence tagging is to predict tags $T = t_1, t_2, \ldots, t_n$ associated with each input. Part-of-speech tagging is a type of sequence tagging problem, where X is an input sequence of words and T is the output sequence of predicted tags for each input word. The most common neural architecture for sequence tagging uses a simple RNN followed by a softmax classifier.

An example of a sequence tagger using an RNN for named entity recognition is given in Figure 2.1. This applies similarly to part-of-speech tagging, just with a different set of output labels. Word embeddings are input to an RNN one-by-one, producing a sequence of hidden states $H = h_1, h_2, \ldots h_n$, one for each word. The hidden states are then transformed by a feedforward network $t_k = feedforward(h_k)$ into a one-hot representation, meaning that every vocabulary item has a corresponding vector dimension. If the output in a particular dimension is the highest, then that means the network predicts that vocabulary item with the highest certainty.

2.3 Morphological Tagging

UD provides a set of morphological features associated with each word, e.g., `Mood=Ind|Tense=Past|VerbForm=Fin`. In a sense, this literal string can be viewed as an example of a "morphology tag." Assuming all tags are ordered according to some deterministic scheme (such as alphabetical), then there will be only one correct morphology tag per word in a sentence. From this perspective, morphological analysis with respect to UD can be thought of as a sequence tagging problem, and

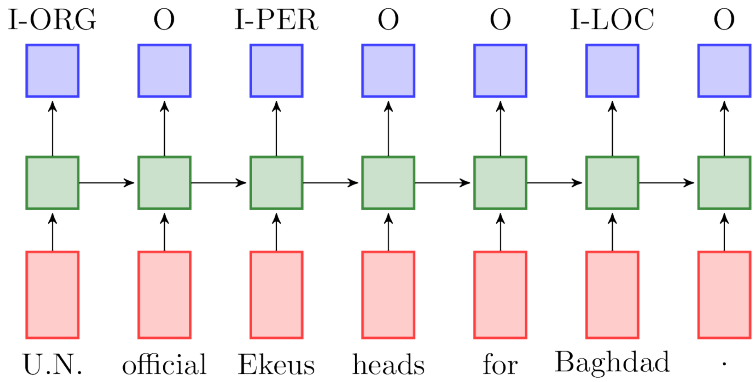

Figure 2.1: An example of sequence tagging applied to named entity recognition using a recurrent neural network [Sterbak, 2017]. Embeddings (red) of words are input into an RNN unit (green) sequentially, producing hidden states (blue). The hidden states are decoded by a feedforward layer followed by a softmax activation function.

so the neural sequence tagging approach can be used identically to part-of-speech tagging, swapping out the output prediction vocabulary.

However, it is quite apparent that this approach does not take into account any of the underlying subcategorical features, but treats all combinations as a whole. This can make it difficult for a model to implicitly know that the difference between two morphological tags can be a much simpler change in one of the morphological features rather than a change of multiple morphological features. Therefore, one can make a small modification and factorize the morphological tag into several dimensions. For instance, there could be separate "mood," "tense," and "verb form" aspects of each word that can be predicted independently of each other. One can either use separate sequence tagger networks to predict each of these subcategories, or share the encoder RNN and calculate multiple feedforward decoders on the hidden states. These approaches to morphological analysis can be seen in works such as Inoue et al. [2017], Kondratyuk et al. [2018].

2.4 Lemmatization

One can view lemmatization as a machine translation problem. Given a sequence of characters of the word form, the goal is to generate the sequence of characters of the lemma. However, we will encounter problems if we try to treat each word form independently from each other, as certain word forms need to be disambiguated before applying the appropriate lemmatization rule. Contextualized embeddings work suitably for this task. Bergmanis and Goldwater [2018] try this approach using a sequence-to-sequence architecture to decode lemmas charcter by character, encoding the context of a sentence by using bidirectional RNNs on the input word embeddings of a sentence. One major downside to this approach is that there is considerable cost in generating lemmas. An RNN decoder generates output one character at a time, slowing down considerably over more traditional non-neural lemmatization models. In addition, most word forms are their inflected lemma, meaning that the majority of the time will be spent doing no-ops and would therefore be wasteful to decode what is essentially a non-operation character by character. And for particularly long tokens like URLs, this can pose a challenge for sequence-to-sequence architectures, as this becomes increas-

ingly more difficult for RNNs the longer the character sequence becomes.

An alternative to the machine translation approach is to treat lemmatization as a sequence tagging problem like what is shown above. Instead of predicting sequences of characters, another possibility is to predict the operations that transform the word form into the lemma. Müller et al. [2015] experiment with this approach, precomputing edit trees that each represent a sequence of character operations that transform the input word into the lemma. For each word-lemma pair, they compute a hierarchical representation (tree) consisting of finding the longest common substring and then recursively model the prefix and suffixes as sub-trees whose spans correspond to which substring is being considered. Leaf nodes define rules to explicitly add, change, or delete characters. Once computed, each edit tree can be considered a "tag" for a given word in a sentence. By predicting the tag, the edit rules defined by the edit tree "tag" can be applied to the input word form to produce the lemma.

2.5 Dependency Parsing

There have been many models proposed for dependency parsing [Chen and Manning, 2014, Zhou et al., 2015, Dyer et al., 2015], but one that has consistently demonstrated state-of-the-art performance in recent years is the graph-based biaffine attention parser developed by Dozat and Manning [2016], Dozat et al. [2017]. An illustration of the biaffine dependency parser is given in Figure 2.2.

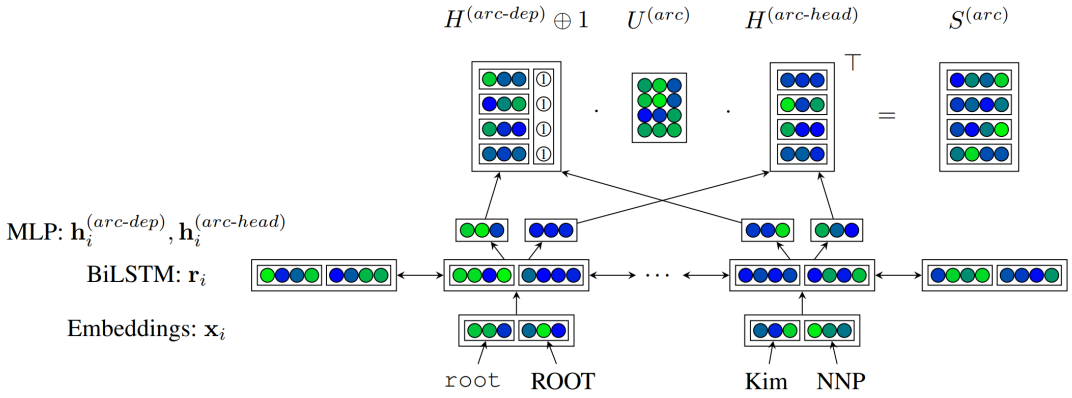

Figure 2.2: The biaffine graph-based dependency parser developed by Dozat and Manning [2016].

As seen in standard recurrent architectures described previously, all input word embeddings in a sentence are passed through bidirectional LSTMs before being decoded into parse trees. In addition to word embeddings, the authors also experiment with concatenating part-of-speech tag embeddings alongside words to enrich the source information with helpful classes for each word. The hidden states of the LSTM are projected through arc-head and arc-dep feedforward layers, which are combined using bilinear attention to produce a probability distribution of arc heads for each word. The bilinear attention can be thought of as a weight matrix relating the strength of a connection between any two input vectors. More precisely, the network computes a vector of arc scores $\boldsymbol{s}_i$ such that

$$\boldsymbol{s}_i^{(arc)} = H^{(arc-head)} U^{(arc)} \boldsymbol{h}_i^{(arc-dep)} + H^{(arc-head)} \boldsymbol{b} \tag{2.1}$$

where $\boldsymbol{s}_i$ is the hidden output produced by the LSTM at token position i, $U^{(arc)}$ is a matrix computing a bilinear matching operation between projected (feedforward) LSTM outputs $H^{(arc-head)}$ and projected LSTM outputs $\boldsymbol{h}_i^{(arc-dep)}$, and bias vector $\boldsymbol{b}$. In simple terms, matching scores are computed between all possible combinations of head-dependent pairs using the bilinear operation, producing probabilities that indicate the likelihood of a word being a dependent of another head word. To produce each label, a similar bilinear operation is calculated with respect to the computed arc probabilities and predicted labels for each arc $\boldsymbol{s}_i^{(arc-label)}$.

To produce dependency trees from these probabilities, one can try to find an optimal spanning tree of the complete graph of probabilities that results in the most probable dependency tree. The simplest approach would to greedily decode each tree, iteratively finding the largest arc-label probabilities and progressively eliminating alternative connections. However, this does not guarantee a valid dependency tree, as there may be cycles in the resulting graph. Dozat and Manning [2016] use a strategy of backtracking when identifying cycles and continuing to iteratively find the best greedy parse. While not optimal, this simple and efficient approach can nearly reach the performance of more sophisticated methods.

A more advanced method would be to take the approach of using the Chu-Liu/Edmonds algorithm [Chu, 1965, Edmonds, 1967] to find the optimal spanning trees from the output probabilities. While also greedy, the algorithm uses more clever delayed backtracking techniques to elminiminate suboptimal combinations and always finds the optimal solution. The algorithm recursively performs two operations, contraction, which identifies cycles, and expansion, which finds the best path to a given node.

2.6 Multi-Task Learning with LemmaTag

LemmaTag [Kondratyuk et al., 2018] experiments with *multi-task learning* by simultaneously predicting lemmas and morphology tags. The network shares the parameters of an encoder, which produces contextualized word representations with bidirectional RNNs (BiRNNs or BRNNs) as seen in Section 2.2. The morphology tags are predicted like standard part-of-speech tags, while the lemmas are predicted in a sequence-to-sequence architecture like Bergmanis and Goldwater [2018].

Both lemmatization and morphology tags require context-sensitive awareness to disambiguate words with the same form but different syntactic or semantic features and behavior. Furthermore, lemmatization of a word form can benefit substantially from the information present in morphological tags, as grammatical attributes often disambiguate word forms using context [Müller et al., 2015]. Results indicate that multi-tasking with several related morpho-syntactic tasks can benefit the performance of all involved tasks. By having the network perform multiple tasks, the idea is to allow it to generate more generalized representations of text, as the network parameters must accommodate calculations that produce representations that perform well on all tasks.

To better understand the effect multi-tasking has on neural networks, we add two additional sequence predictors to the encoder to jointly predict morphological features (feats), universal part-of-speech (upos), treebank-specific part-of-speech (upos) and lemmas. See Figure 2.3 for an ablation that compares between the network predicting all tasks jointly and the network performing just one of the four tasks. The blue lines

indicate the sequence accuracy scores calculated on the fully joint model, and the other colors indicate each task performed separately. The plot shows that overall, multi-tasking improves the evaluation accuracy of these related tasks over just having four separate models.

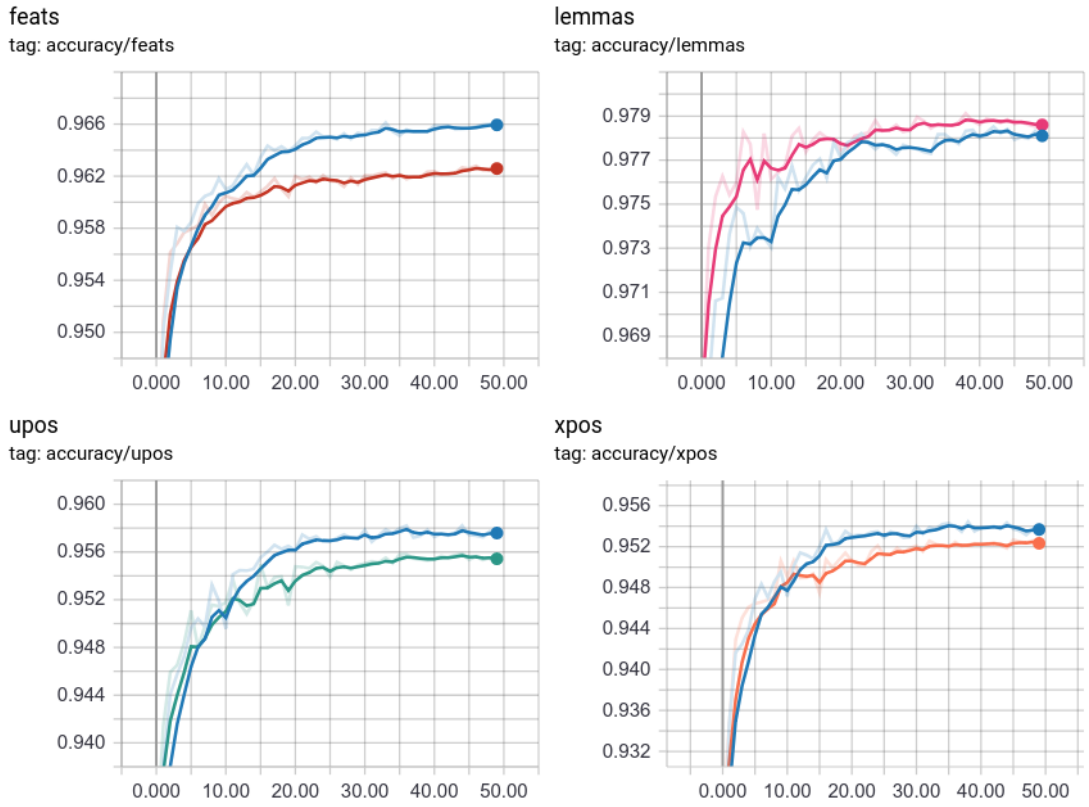

Figure 2.3: A plot of multi-task development scores predicted using a recurrent neural network on the UD English EWT dataset. Blue lines indicate all tasks are performed jointly. All other colors indicate the task was performed separately.

LemmaTag additionally combines its initial word vectors with *character embeddings*. The input characters corresponding to each word are represented as character embeddings, which are fed through bidirectional RNNs similar to the word vectors. The final hidden states are summed together with the word vectors to produce character-aware embeddings, which have been shown to improve the performance of networks modeling syntax and especially morphology [Santos and Zadrozny, 2014, Ling et al., 2015, Ballesteros et al., 2015, Kim et al., 2016, Heigold et al., 2017].

A natural extension of LemmaTag would be to use external sources of data to improve prediction accuracy. From what was introduced in Chapter 1, one could think of using contextualized embeddings produced by BERT and inserting them into LemmaTag, which would almost certainly boost performance. However, what is more uncertain is the effect multilinguality has on model performance.

We perform a preliminary experiment on LemmaTag to show that BERT boosts performance, even on languages it was never pretrained on, seen in Table 2.1. We use a BERT multilingual model provided by Devlin et al. [2018], pretrained on a collection of text extracted from Wikipedia containing 104 languages. We modify LemmaTag by concatenating embeddings produced by BERT with the word embeddings used by LemmaTag and inputting them into the network. Then we train LemmaTag on the UD

Kurmanji MG treebank, which is not one of the 104 languages used for pretraining BERT, and we train the network both with and without BERT. We ensure we do not fine-tune the BERT network but instead freeze the weights.

	LEMMATAG	LEMMATAG + BERT
Lemma	55.22	57.01
XPOS	42.91	60.89
UPOS	40.43	62.73
UFeats	39.56	44.20

Table 2.1: Accuracies of UD tasks on Kurmanji comparing between two network configurations: a recurrent network and a recurrent network with contextual embeddings provided by multilingual BERT.

The results are clear that despite never observing Kurmanji, BERT somehow possesses multilingual information that nonetheless can help a network improve its evaluation performance. This observation makes it apparent that multilingual BERT can possibly enable the development of a massively multilingual syntactic parser, and can further provide improved performance over using monolingual BERT.

2.7 Multi-Task Learning with UDPipe Future

Of the entries submitted to the CoNLL 2018 Shared Task, UDPipe Future developed by Straka [2018] can be thought of as an extended version of LemmaTag supporting the prediction of all UD tasks. This will form the basis for a multilingual a singular model that, similar to how Universal Dependencies provides annotations universal across all languages, will provide predictions of Universal Dependencies universally across all supported languages.

Similar to LemmaTag, UDPipe Future uses a bidirectional recurrent network to provide contextual embeddings of all words in an input sentence, and enriches this information with character-level embeddings produced by additional BRNNs. The network also predicts UPOS, XPOS, and UFeats as simple part-of-speech tags. But the major difference between the networks is in lemmatization and dependency parsing. UDPipe Future uses a lemma tagging approach as opposed to the sequence-to-sequence approach seen in LemmaTag and Bergmanis and Goldwater [2018]. UDPipe Future predicts edit scripts that define sequences of character operations to transform the word form into the lemma. These operations are calculated using Levenshtein distance to find the minimum operations required for string transduction. Finally, UDPipe Future uses the dependency parser by Dozat and Manning [2016] attached to the BRNN encoder to predict the UD dependency trees.

2.8 Metrics for Scoring UD Predictions

When predicting UD annotations, it is important to use standard metrics for determining the quality of parsed outputs. To that end, we define the metric calculations used

for scoring:

UPOS, XPOS, UFeats, Lemma We use accuracy scores for all tasks that have a 1-1 correspondence between the input words and output strings. The basic calculation is defined as the number of correct predictions divided by the total number of predictions, over the entire dataset.

UAS, LAS As dependency trees have structural information with respect to the entire sentence, they require slightly more sophisticated evaluation. An unlabeled attachment score (UAS) is defined as the percentage of words that have the correct head, i.e., the number of words having the correct head in the dataset divided by the total number of words in the dataset. This is possible due to the dependency tree constraint specifying each word must be the dependent of exactly one other word in the sentence. The labeled attachment score (LAS) is similar, except only counting correct cases when the word has both the correct head and the correct label for that head.

All scores are *macro-averaged*, meaning that we do not calculate intermediate averages over sentences, but accumulate counts over the entire treebank before averaging.

3. The UDify Model

In this chapter, we provide a detailed overview of UDify, a single multilingual multi-task model that annotates Universal Dependencies on text in any of 75 supported languages. Our work uses the AllenNLP library built for the PyTorch framework. Code for UDify and a release of the fine-tuned BERT weights are available at `https://github.com/hyperparticle/udify`.

3.1 Design Considerations

UDify is a continuation of Kondratyuk et al. [2018], which presents a multi-task network jointly processing part-of-speech tags and lemmas. This thesis expands the joint model, extending to all Universal Dependencies Tasks, replacing the recurrent components with a properly regularized self-attention encoder network, and training the network on an expanded multilingual dataset.

This new UDify model is based on UDPipe Future, a winner of the CoNLL 2018 shared task, as it introduces a simple model similar to LemmaTag which provides a natural extension to all UD tasks. We perform three primary modifications to UDPipe Future [Straka, 2018]:

1. We replace all recurrent components with a pretrained multilingual BERT self-attention network which provides contextual embeddings and introduce a novel layer-wise attention for each task.

2. We apply a heavy amount of regularization to BERT, including dropout, input masking, weight freezing, discriminative fine-tuning, and layer dropout.

3. We train UDify on the entirety of UD by concatenating *all* available training sets together, as seen in McDonald et al. [2011].

The final outcome of the UDify model would not be possible without the recent advances in contextualized word representations, the development of BERT, the release of a multilingual BERT model, and the fact that the Universal Dependencies annotations are truly universal across all languages. Devlin et al. [2018] released several versions of the BERT model, including a multilingual model pretrained on the entirety of the top 104 resourced languages of Wikipedia. Conveniently, the 104 languages pretrained on the multilingual model nearly completely overlap with languages supported by the Universal Dependencies treebanks. This connection spurred the development of a method to exploit this model for state-of-the-art Universal Dependency parsing using as simple of an approach as possible. We leverage the provided BERT base multilingual cased pretrained model[1], with a self-attention network of 12 layers, 12 attention heads per layer, and hidden dimensions of 768. This particular model uses a wordpiece tokenizer [Wu et al., 2016] which segments all text into unnormalized sub-word units.

We note that the architecture of UDify could be enhanced with more advanced recent methods in sequence modeling and dependency parsing which would boost performance, but this thesis opts to show that these additional features are unnecessary

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

to reach state-of-the-art performance. We desired to not use any recurrent structures like LSTMs so that the model can be highly parallelizable, and show that recurrent structures are not necessary for good performance on the sequence modeling of small datasets, which is something that has been shown only very recently [Strubell et al., 2018, Kitaev and Klein, 2018a, Baevski et al., 2019]. We also desired explicitly not to include any treebank or language-specific components like language identifiers in the input or language classifiers in the output, which would encourage the model to create representations that either recognize the languages automatically or create language-invariant generalizations.

While the architecture is based on UDPipe Future, the multilingual modeling idea is most similar to the Uppsala system for the CoNLL 2018 Shared Task [Smith et al., 2018]. Uppsala combines treebanks of one language or closely related languages together over 82 treebanks and parses all UD annotations in a multi-task pipeline architecture. This approach reduces the number of models required to parse each language while also showing results that are no worse than training on each treebank individually, and in especially low-resource cases, significantly improved.

To the best of our knowledge, this is the first massively multilingual self-attention network processing Universal Dependencies capable of meeting or exceeding state-of-the-art accuracy (we note that this thesis was under development, Artetxe and Schwenk [2018] have also recently also demonstrated a highly scalable approach to produce multilingual embeddings for natural language inference with up to 93 languages using more traditional BiLSTMs). And as a nice side-effect, we are also able to analyze the self-attention heads of BERT to see if the representations it has learned resemble that of the dependency trees it is tasked to predict.

3.1.1 Training on all Universal Dependencies Training Data Simultaneously

To enable UDify to produce one model capable of processing any UD language, we take the simple approach of concatenating all training data together to form one large dataset, similar to McDonald et al. [2011]. Before each epoch, we shuffle all sentences and feed mixed batches of sentences to the network, where each batch may contain sentences from any language or treebank. Surprisingly, this approach is all that is required, and no special preprocessing other than BERT's wordpiece tokenizer is necessary. This can be owed to the fact that the BERT model has already implicitly learned to identify each language during pretraining.

However, it is very important to consider just how large the combined Universal Dependencies corpus is. If we concatenate all training, development, and test data across all treebanks, we can reveal an enormous number of unique tokens. Table 3.1 displays a list of vocabulary sizes, showing that UD treebanks possess nearly 1.6 million unique tokens combined across more than 12 million sentences. Due to memory limitations, embedding tables typically do not exceed 200,000 unique words, so this vocabulary must be reduced in size. The simplest approach would be to replace the tokens that are not in the top most frequent word list with a special unknown token `[UNK]`, but since we are dealing with multilingual data, we would inadvertently exclude the tokens of many low-resource languages.

To sidestep the problem of a ballooning vocabulary, the UDify model uses BERT's wordpiece tokenizer directly. We observe that BERT's tokenizer is originally intro-

TOKEN	VOCAB SIZE
Word Form	1,588,655
BERT Wordpieces	119,547
UPOS	17
XPOS	19,126
UFeats	23,974
Lemmas (tags)	109,639
Deps	251

Table 3.1: Vocabulary sizes of words and tags over all of UD v2.3, with a total of 12,032,309 word tokens and 668,939 sentences.

duced precisely to circumvent this issue by breaking words up into smaller sequences of characters. This drastically reduces the vocabulary size to a manageable 120,000 tokens, and eliminates the need to keep a separate embedding table. UD expects predictions to be along word boundaries, so we take the simple approach of applying the tokenizer to each word using UD's provided word segmentation. For prediction, we use the outputs of BERT corresponding to the first wordpiece per word, ignoring the rest. We found there was no discernable difference in using the first wordpiece, last wordpiece, taking an average, or taking the max wordpiece with respect to evaluation performance. Kitaev and Klein [2018b] report similar results when fine-tuning BERT for dependency parsing.

In addition, the XPOS annotations are not universal across languages, or even across treebanks. The part-of-speech tags may have a small number of coarse categories like the 17 different types of UPOS tags, or the nearly 1,500 tags defined by the Prague Dependency Treebank which encode more fine-grained morphological characteristics. Because each treebank can possess a different annotation scheme for XPOS which can slow down inference, we omit training and evaluation of XPOS from our experiments. However, we note that it may be possible to include a treebank identifier in the input that provides the model with enough information to infer which XPOS annotation to produce. But because this requires knowledge of the underlying language and treebank, we leave this out for future work to keep the model simple.

One caveat we had prior to developing the model was that despite having "Universal" annotations, treebanks of one language usually have different annotators whose text come from different domains, and so may not completely agree with each other. However, as shall be seen in Chapter 4, this discrepancy must be relatively small (or perhaps the network can easily identify which tricky sentences belong with which treebank). The model frequently performs just as well or even better on multi-treebank training as on single treebanks.

3.2 Layer Attention

The authors of BERT have observed that when fine-tuning BERT, combining the output of the last several layers is more beneficial for the downstream tasks than just using the last layer [Devlin et al., 2018]. Table 3.2 illustrates this point, showing F1 scores when fine-tuning BERT on the CoNLL 2003 shared task on named entity recognition.

LAYERS	DEV F1
Finetune All	96.4
First Layer (Embeddings)	91.0
Second-to-Last Hidden	95.6
Last Hidden	94.9
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1
Sum All 12 Layers	95.5

Table 3.2: Ablation comparing fine-tuning strategies on various layers of BERT. These results are taken from the original paper of Devlin et al. [2018].

This leads to the interesting observation that not only do different layers encode different information, the different layers also encode non-overlapping information that can be combined to boost evaluation performance. This is an observation also echoed by the authors of ELMo who found that combining all layers of a language model in some fashion is superior to just using the last layer [Peters et al., 2018].

Instead of restricting the model to any subset of layers, we devise a simple layer-wise dot-product attention where the network computes a weighted sum of all intermediate outputs of the 12 BERT layers using the same weights for each token. This is very similar to how ELMo combines its own RNN layers to produce its contextual embeddings. More formally, let $\boldsymbol{\alpha}_i$ be a trainable scalar for BERT embeddings $\boldsymbol{B}_{ij}$ at layer i with a token at position j, and let β be a trainable scalar. We compute contextual embeddings $\boldsymbol{e}$ such that

$$\boldsymbol{e}_j = \beta \sum_i \boldsymbol{B}_{ij} \cdot \mathrm{softmax}(\boldsymbol{\alpha})_i \tag{3.1}$$

This corresponds to a normalized weighted sum of the embeddings of all BERT layers, scaled by β.

3.3 Model Architecture

For predicting UD annotations, we employ a multi-task network like UDPipe Future [Straka, 2018], but with all embedding, encoder, and projection layers replaced with BERT. The remaining parts include layer attention and the prediction layers for each task detailed below.

See Figure 3.1 for an architecture diagram. A given sentence is passed through BERT's wordpiece tokenizer, possibly breaking up words into smaller sequences of characters. If a word is broken up into more than one subunits, the position of the embeddings corresponding to the first subunit is used to represent the word. The whole tokenized sentence is passed through BERT by using embedding lookups and successively applying self-attention at each of the layers. Once the outputs of all 12 layers are calculated, task-specific layer attention successively computes a weighted average of the layers of BERT for a total of 4 separate averages. Each averaged result is then passed to a classifier which predicts one of 4 tasks.

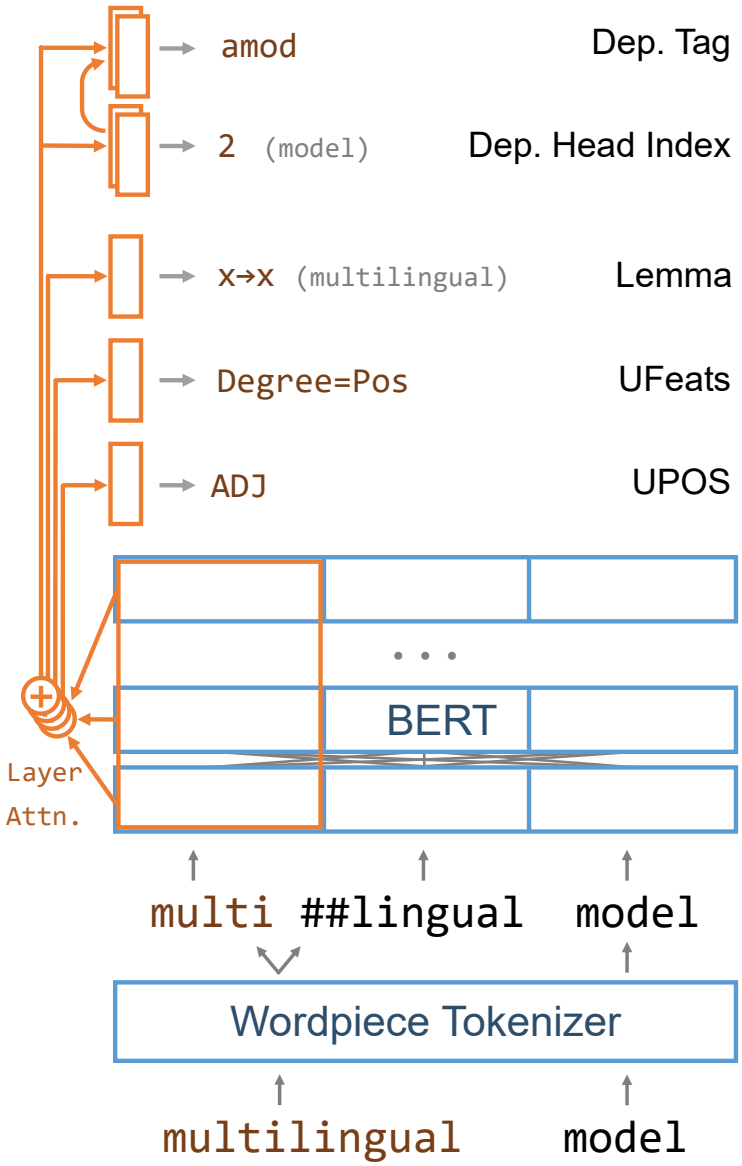

Figure 3.1: An illustration of the UDify network architecture with task-specific layer attention, inputting word tokens and outputting UD annotations for each token.

3.3.1 Model Tasks

A short description of the model tasks are below.

UPOS As is standard for neural sequence tagging, we apply a softmax classifier along each word input, predicting the annotation string.

UFeats Identical to UPOS prediction, we treat each UFeats string as a separate token in the vocabulary. We found this to produce higher evaluation accuracy than predicting each morphological feature separately. Only a small subset of the full Cartesian product of morphological features is valid, eliminating invalid combinations. We could have also factored the morphological tags to also predict each individual morphological dimension as in Kondratyuk et al. [2018], but we observe only marginal improvement for training a larger model requiring more training time, so we did not take this approach.

Lemmas Similar to Chrupała [2006], Müller et al. [2015], we reduce the problem of lemmatization to a sequence tagging problem by predicting a class representing an edit script, i.e., the sequence of character operations to transform the word form to the lemma. To precompute the tags, we first find the longest common substring between the form and the lemma, and then compute the shortest edit script converting the prefix and suffix of the form into the prefix and suffix of the lemma using the Wagner–Fischer algorithm [Wagner and Fischer, 1974]. Upon predicting a lemma edit script, we apply the edit operations to the word form to produce the final lemma. We take this tagging approach over a sequence-to-sequence approach like Bergmanis and Goldwater [2018], as it requires much less computation time to predict lemmas while also not hurting evaluation performance significantly.

Deps We use the graph-based biaffine attention parser developed by Dozat and Manning [2016], Dozat et al. [2017], replacing the bidirectional LSTM layers with BERT. The final embeddings are projected through arc-head and arc-dep feedforward layers, which are combined using biaffine attention to produce a probability distribution of arc heads for each word. We then decode each tree with the Chu-Liu/Edmonds algorithm [Chu, 1965, Edmonds, 1967].

3.3.2 Extremely Long Sentences

BERT limits its positional encoding to 512 wordpieces, causing some sentences in UD to be too long to fit into the model. We use a sliding window approach to break up long sentences into windows of 512 wordpieces, overlapping each window by 256 wordpieces. After feeding the windows into BERT, we select the first 256 wordpieces of each window and any remaining wordpieces in the last window to represent the contextual embeddings of each word in the original sentence.

3.4 Regularization Strategies

UDify employs several strategies for fine-tuning BERT for Universal Dependency prediction, and we find that regularization is absolutely crucial for producing a high-

scoring network. Without this, the network would not be able to surpass the UDPipe Future baseline and would quickly overfit to the training data. We observed in preliminary experiments that this would likely produce a model that on average evaluates 10% lower in score than typical state-of-the-art UD parsers.

3.4.1 Layer Dropout

Most of the time the addition of layer attention would not be very useful on its own, as the attention weights would be scaled such that the final output would be effectively equivalent to taking the output of the final one or two layers. To prevent the UD classifiers from overfitting to the information in any single layer, we devise layer dropout, where at each training step, we set each layer-scaling parameter $\boldsymbol{\alpha}_i$ to $-\infty$ with probability 0.1. Due to the properties of the softmax function, this effectively redistributes probability mass to all other layers, forcing the network to incorporate the information content of all BERT layers. We compute layer attention per task, using one set of $\boldsymbol{\alpha}, \beta$ parameters for each of the classifiers of UPOS, UFeats, Lemmas, and Deps.

3.4.2 Transfer Learning with ULMFiT

The ULMFiT strategy defines several useful methods for fine-tuning a network on a pretrained language model [Howard and Ruder, 2018]. We apply the same methods, with a few minor modifications.

We split the network into two parameter groups, i.e., the parameters of BERT and all other parameters. We apply discriminative fine-tuning, setting the base learning rate of BERT to be $5e^{-5}$ and $1e^{-3}$ everywhere else. We also use gradual unfreezing to freeze the BERT parameters for the first epoch to increase training stability, and then unfreeze for all remaining epochs.

While ULMFiT recommends decaying the learning rate linearly after a linear warm up, we found that this is prone to training divergence in self-attention networks, introducing vanishing gradients and underfitting. Instead, we apply an inverse square root learning rate decay with linear warmup (Noam) seen in training Transformer networks for machine translation [Vaswani et al., 2017].

This allows the classifiers to converge more quickly while keeping the BERT learning rate low enough to prevent training divergence. This not only stabilizes training, but we have also observed higher test accuracy when fine-tuning BERT with Noam over slanted triangular.

3.4.3 Input Masking

The authors of BERT recommend not to mask words randomly with `[MASK]` when fine-tuning the network. However, we discovered that masking often reduces the tendency of the classifiers to overfit to BERT. This *word dropout* strategy has been observed in other works showing improved test performance on a variety of NLP tasks [Iyyer et al., 2015, Bowman et al., 2016, Clark et al., 2018, Straka, 2018]. The intuition is that masking words forces the network to rely on the context of surrounding words to accurately classify a word rather than always relying on the lexical entry itself. This also makes the model much less likely to overfit to any particular sentence, as it is never able to see the entire contents of any sentences, at least not directly.

3.4.4 Dropout

Despite using all the regularization strategies above, we still observe overfitting and must apply more aggressive techniques. To further regularize the network, we also increase the attention and hidden dropout rates of BERT from 0.1 to 0.2, and we also apply a dropout rate of 0.5 to all BERT layers before computing layer attention for each of the four tasks and applying a layer dropout with probability 0.1. We increase the masking probability of each wordpiece from 0.15 to 0.2.

3.5 Hyperparameters & Training Details

HYPERPARAMETER	VALUE
Dependency tag dimension	256
Dependency arc dimension	768
Optimizer	Adam
β_1, β_2	0.9, 0.99
Weight decay	0.01
Label smoothing	0.03
Dropout before layer attention	0.5
BERT dropout	0.2
Mask probability	0.2
Layer dropout	0.1
Batch size	32
Epochs	80
Base learning rate	$1e^{-3}$
BERT learning rate	$5e^{-5}$
Learning rate warmup steps	8000
Gradient clipping	5.0

Table 3.3: A summary of model hyperparameters.

Upon concatenating all training sets, we shuffle all the sentences, bundle them into batches of 32 sentences each, and train UDify for a total of 80 epochs before stopping. We hold the learning rate constant until we unfreeze BERT in the second epoch, where we and linearly warm up the learning rate for the next 8,000 batches and then apply inverse square root learning rate decay for the remaining epochs. For the dependency parser, we use feedforward tag and arc dimensions of 300 and 800 respectively. We apply a small weight decay penalty of 0.01 to ensure that the weights remain small after each update. For optimization we use the Adam optimizer and we compute softmax cross entropy loss to train the network. We use a default β_1 value of 0.9 and lower the β_2 value from the typical 0.999 to 0.99. The reasoning is to increase the decay rate of the second moment in the Adam optimizer to reduce the chance of the optimizer being too optimistic with respect to the gradient history. We clip the gradient updates to a maximum L2 magnitude of 5.0. A summary of hyperparameters can be found in Table 3.3.

To speed up training, we employ bucketed batching, sorting all sentences by their length and grouping similar length sentences into each batch. However, to ensure that

most sentences do not get grouped within the same batch, we fuzz the lengths of each sentence by a maximum of 10% of its true length when grouping sentences together.

With all these regularization strategies and hyperparameter choices combined, we are able to fine-tune BERT for far more epochs before the network starts to overfit, i.e., 80 as opposed to around 10. Even so, we believe even more regularization can improve test performance.

The final multilingual UDify model was trained over approximately 25 days on an NVIDIA GTX 1080 Ti taking an average of 8 hours per epoch. We use half-precision (fp16) training to be able to keep the BERT model in memory. One notable aspect of training is that while we observed the model start to level out in validation performance at around epoch 30, the model continually made small, incremental improvements over each subsequent epoch, resulting in far higher scores than if the model training was terminated early. This can be partially attributed to the decaying learning rate explained in Section 3.4.2.

Due to the high training times, we must pick and choose a small number of training experiments the most relevant and useful results. Prior to developing the final model, we conducted fine-tuning experiments on pairs of languages to find a set of hyperparameters that worked best for multilingual learning. After this, we gradually scaled up training to 3 languages, 5 languages, 15 languages, and then finally the model presented above. We had high doubts, and wanted to see where the limit was in multilingual training. We were surprised to find that this simple training scheme was able to scale up so well to all UD treebanks.

4. Experiments and Results

In this chapter, we evaluate UDify with respect to every test set in each treebank, showing strong performance when compared to UDPipe Future. Then we analyze the internals of UDify to see what the model has learned and what has changed from the original BERT pretrained model. We then discuss why pretrained self-attention networks excel in Universal Dependency parsing.

We do not directly reference metrics from the CoNLL 2018 Shared Task, as the tables of results do not assume gold word segmentation and may not provide a fair comparison. Instead, we retrained the open source UDPipe Future model using gold segmentation and report results here due to its architectural similarity to UDify and its strong performance.

As there are too many results to discuss all at once, we display a salient subset of scores below and compare them with UDPipe Future. To see the full contents of all training results, we display them at the end of Section 4.5.

4.1 Datasets

For all experiments, we use the full Universal Dependencies v2.3 corpus available on LINDAT [Nivre, 2018]. We assume the gold segmentation of tokens provided by UD. Due to licensing restrictions, we omit the evaluation of datasets that do not have their training annotations freely available, i.e., Arabic NYUAD (ar_nyuad), English ESL (en_esl), French FTB (fr_ftb), Hindi English HEINCS (qhe_heincs), and Japanese BCCWJ (ja_bccwj).

4.2 Training on All Treebanks vs. One Treebank

To provide a better comparison that reveals what UDify has learned with respect to multilinguality, we select 6 high-resource and 5 low-resource languages and perform two additional experiments.

Aside from training on all treebanks, we also compare with performing the same setup on just one treebank. A comparison of scores would reveal if languages with a single treebank will improve in score and therefore incorporate useful multilingual information when transitioning to the multilingual training setup.

In preliminary experiments, we have noticed that multilingual fine-tuning followed by a second round of fine-tuning on a single treebank can possibly boost evaluation scores. Once a model has learned an adequate multilingual representation for UD parsing, we can fine-tune again on just one treebank. The reasoning is that while multilingual pretraining can help, the distribution of data it is trained on is very different from the test set we are evaluating on. The network can be misled by other languages, especially if they are distant from most other languages, and so we can fine-tune these representations to focus the most on our language of choice.

4.3 Effect of Syntactic Fine-Tuning on BERT

We show scores of UPOS, UFeats (FEATS), and Lemma (LEM) sequence accuracies, along with unlabeled and labeled attachment scores (UAS, LAS) evaluated using the offical CoNLL 2018 Shared Task evaluation script[1]. For comparisons across different model configurations, we bold the highest evaluation scores for each treebank and across each metric. Results for a salient subset of high-resource and low-resource languages are shown in Table 4.1 and Table 4.2 respectively, with a comparison between UDPipe Future and UDify fine-tuning on all languages. In addition, the table compares UDify with fine-tuning on a single language, or both (fine-tuning multilingually, then fine-tuning on the language with the saved multilingual weights) to provide a reference point for multilingual influences on UDify.

TREEBANK	MODEL	UPOS	FEATS	LEM	UAS	LAS
Czech PDT (cs_pdt)	UDPipe	99.18	97.23	**99.02**	93.33	91.31
	Lang	99.18	96.87	98.72	94.35	92.41
	UDify	99.18	96.85	98.56	94.73	92.88
	UDify+Lang	**99.24**	**97.44**	98.93	**95.07**	**93.38**
German GSD (de_gsd)	UDPipe	94.48	90.68	**96.80**	85.53	81.07
	Lang	94.77	91.73	96.34	87.54	83.39
	UDify	94.55	90.65	94.82	87.81	83.59
	UDify+Lang	**95.29**	**91.94**	96.74	**88.11**	**84.13**
English EWT (en_ewt)	UDPipe	96.29	97.10	**98.25**	89.63	86.97
	Lang	**96.82**	**97.27**	97.97	**91.70**	**89.38**
	UDify	96.21	96.17	97.35	90.96	88.50
	UDify+Lang	96.57	96.96	97.90	91.55	89.06
Spanish AnCora (es_ancora)	UDPipe	**98.91**	**98.49**	**99.17**	92.34	90.26
	Lang	98.60	98.14	98.52	92.82	90.52
	UDify	98.53	97.84	98.09	92.99	90.50
	UDify+Lang	98.68	98.25	98.68	**93.35**	**91.28**
French GSD (fr_gsd)	UDPipe	97.63	**97.13**	**98.35**	90.65	88.06
	Lang	**98.05**	96.26	97.96	92.77	90.61
	UDify	97.83	96.59	97.48	**93.60**	**91.45**
	UDify+Lang	97.96	96.73	98.17	93.56	91.45
Russian SynTagRus (ru_syntagrus)	UDPipe	**99.12**	**97.57**	**98.53**	93.80	92.32
	Lang	98.90	96.58	95.16	94.40	92.72
	UDify	98.97	96.35	94.43	94.83	93.13
	UDify+Lang	99.08	97.22	96.58	**95.13**	**93.70**

Table 4.1: Test set scores for a subset of high-resource languages in comparison to UDPipe Future, with 3 UDify configurations: **Lang**, fine-tune on the treebank. **UDify**, fine-tune on all UD treebanks combined. **UDify+Lang**, fine-tune on the treebank using BERT weights saved from fine-tuning on all UD treebanks combined.

[1] https://universaldependencies.org/conll18/evaluation.html

On average, UDify reveals a strong set of results that are comparable in performance with the state-of-the-art in parsing UD annotations. UDify excels in dependency parsing, exceeding UDPipe Future by a large margin especially for low-resource languages.

Echoing results seen in Smith et al. [2018], UDify also shows strong improvement leveraging multilingual data. In low-resource cases, fine-tuning BERT on all treebanks can be far superior to fine-tuning monolingually. A second round of fine-tuning on an individual treebank using UDify's BERT weights can improve this further, especially for treebanks that underperform the baseline. However, for languages that already display strong results, we typically notice worse evaluation performance across all the evaluation metrics. This indicates that multilingual fine-tuning really is superior to single language fine-tuning with respect to these high-performing languages, showing improvements of up to 20% reduction in error.

TREEBANK	MODEL	UPOS	FEATS	LEM	UAS	LAS
Belarusian HSE (be_hse)	UDPipe	93.63	73.30	87.34	78.58	72.72
	Lang	95.88	76.12	84.52	83.94	79.02
	UDify	**97.54**	**89.36**	85.46	**91.82**	**87.19**
	UDify+Lang	97.25	85.02	**88.71**	90.67	86.98
Buryat BDT (bxr_bdt)	UDPipe	40.34	32.40	58.17	32.60	18.83
	Lang	52.54	37.03	54.64	29.63	15.82
	UDify	**61.73**	**47.86**	**61.06**	**48.43**	**26.28**
	UDify+Lang	61.73	42.79	58.20	33.06	18.65
Upper Sorbian UFAL (hsb_ufal)	UDPipe	62.93	41.10	68.68	45.58	34.54
	Lang	73.70	46.28	58.02	39.02	28.70
	UDify	84.87	48.63	**72.73**	**71.55**	**62.82**
	UDify+Lang	**87.58**	**53.19**	71.88	71.40	60.65
Kazakh KTB (kk_ktb)	UDPipe	55.84	40.40	63.96	53.30	33.38
	Lang	73.52	46.60	57.84	50.38	32.61
	UDify	**85.59**	**65.14**	**77.40**	**74.77**	**63.66**
	UDify+Lang	81.32	60.50	67.30	69.16	53.14
Lithuanian HSE (lt_hse)	UDPipe	81.70	60.47	**76.89**	51.98	42.17
	Lang	83.40	54.34	58.77	51.23	38.96
	UDify	**90.47**	**68.96**	67.83	**79.06**	**69.34**
	UDify+Lang	84.53	56.98	58.21	58.40	39.91

Table 4.2: Test set scores for a subset of low-resource languages in comparison to UDPipe Future, with 3 UDify configurations: **Lang**, fine-tune on the treebank. **UDify**, fine-tune on all UD treebanks combined. **UDify+Lang**, fine-tune on the treebank using BERT weights saved from fine-tuning on all UD treebanks combined.

Interestingly, Slavic languages tend to perform the best with multilingual training. While languages like Czech and Russian possess the largest UD treebanks and do not differ as much in performance from monolingual fine-tuning, evidenced by the improvements over single-language fine-tuning, we can see a large degree of morphological and syntactic structure has transferred to low-resource Slavic languages like

Upper Sorbian, whose treebank contains only 646 sentences. But this is not only true of Slavic languages, as the Turkic language Kazakh (with less than 1,000 training sentences) has also improved significantly.

We also see that despite leveraging the BERT model for monolingual fine-tuning, it does not perform much better than UDPipe, and for low-resource languages typically performs worse. Multilinguality is the core reason we are able to surpass many of the displayed scores.

4.3.1 Overall Performance

To better capture the overall performance of UDify with respect to UDPipe Future, a more comprehensive overview is shown in Table 4.3, comparing different attention strategies applied to UDify. We display an unweighted average of scores over all (89) treebanks with a training set. We do not average all 124 treebanks, as UDPipe Future is not trained multilingually, and so requires a training set for each language it is evaluated on.

MODEL	CONFIGURATION	UPOS	FEATS	LEM	UAS	LAS
UDPipe		**93.76**	**91.04**	**94.63**	84.37	79.76
UDify	Task Layer Attn	93.40	88.72	90.41	**85.69**	**80.43**
UDify	Global Layer Attn	93.12	87.53	89.03	85.07	79.49
UDify	Sum Layers	93.02	87.20	88.70	84.97	79.33

Table 4.3: Ablation comparing the average of scores over all 89 treebanks with a training set: task-specific layer attention, global layer attention for all tasks, and simple sum of layers.

The results show that while UDify is capable of surpassing UPOS, UFeats, and Lemma accuracies for many languages, on the whole it is slightly worse than UDPipe Future. UDify underperforms the most overall with respect to Lemmas and Universal Features, likely due to UDPipe Future additionally using character-level embeddings, while UDify does not.

As explained earlier, character-level modeling helps neural networks to identify patterns in word affixes. Straka [2018] empirically confirms, affixes like prefixes and suffixes convey information about the morphology of a word, and so certain character patterns would help train the network to generalize to better predict each task. Lemmatization and morphological analysis are both heavily related to the surface-level features of words, so this would explain why these scores are the lowest when compared to UDPipe Future, as opposed to higher-level syntactic tasks like UPOS tagging and dependency parsing. To keep the model as simple as possible, we leave character-level processing for future work. But with UPOS, we see the story begin to change, as part-of-speech tags tend to be more contextual in nature, requiring word-sense disambiguation in many cases. And with respect to Deps, the scores reverse, owing to the fact that self-attention networks are excellent in identifying dependencies between words. The next sections will provide more evidence for this claim.

Table 4.3 also shows that layer attention on BERT for each task is beneficial for test performance, much more than using a global weighted average. If we use just a

simple sum of layers or even just the final BERT layer, UDify would not significantly surpass UDPipe Future in any of the given tasks. By allowing the sum to be a trainable weighted sum, we see that this global layer attention produces a small boost in performance. But the largest performance boost is in allowing the different tasks to attend to different layers. The logical conclusion is that each layer encodes distinct information that each task uses uniquely.

4.3.2 Layer Attention Preference

We plot the layer attention weights $\boldsymbol{\alpha}$ after fine-tuning BERT in Figure 4.1, showing a set of weights per task. The weights show that each task prefers the layers of BERT differently. All tasks favor the information content in the last 3 layers, with a tendency to disprefer layers closer to the input.

However, an interesting observation is that for Lemmas and UFeats, the classifier prefers to also incorporate the information of the first 3 layers. This meshes well with the linguistic intuition that morphological features are more closely related to the surface form of a word and rely less on context than other syntactic tasks. Curiously enough, the middle layers are highly dispreferred, meaning that the most useful processing for multilingual syntax (tagging, dependency parsing) occurs in the last 3-4 layers. It appears that BERT's strong contextual capabilities do not manifest themselves until the last few layers.

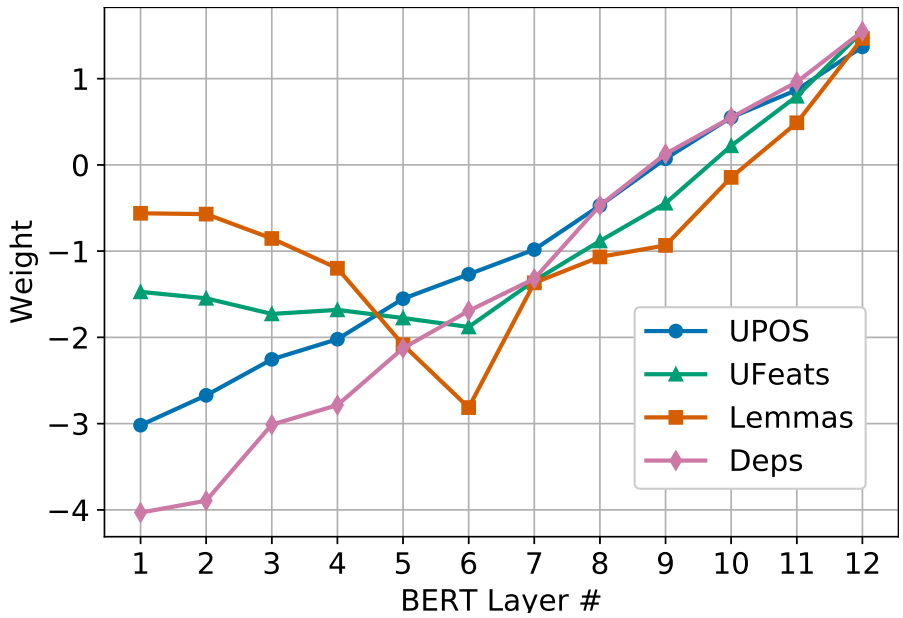

Figure 4.1: The unnormalized BERT layer attention weights $\boldsymbol{\alpha}_i$ contributing to layer i for each task after training. A linear change in weight scales each BERT layer exponentially due to the softmax in Equation 3.1

Figure 4.2 plots the layer attention weights after training the global attention model. We see that the weights closely resemble a sum of the weights of the tasks shown in Figure 4.1. As each task is competing for the same weights, we see that the attention weights compromise on providing the greatest benefit evenly distributed to all tasks. But by providing weights for the individual tasks, they no longer have to compete for which layers to attend to, and can freely extract the information most useful for their task.

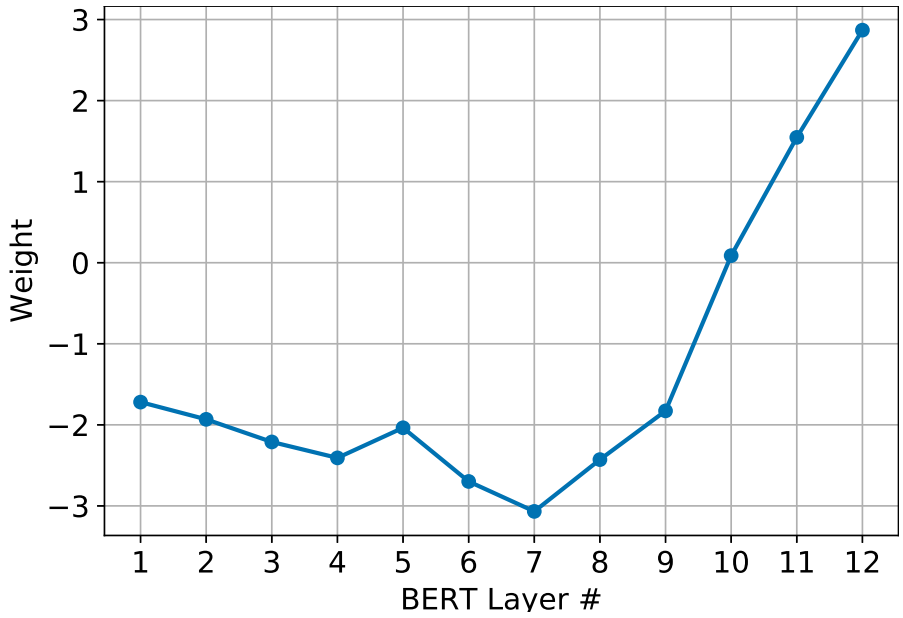

Figure 4.2: The global unnormalized BERT layer attention weights α_i contributing to layer i.

The results released by [Tenney et al., 2019] also agree with the intuition behind the weight distribution above, showing how the different layers of BERT are most beneficial to different tasks using a similar weighting strategy. The layers provide information similar to a traditional NLP pipeline, where low-level tasks like POS tagging are best performed in lower layers and higher layers compute rely on these lower-level tasks to generate high-level abstractions useful for e.g., coreference resolution and semantic role labeling. A per-task weighting of the layers, therefore, can work better because of the rich information generated by these intermediate tasks.

4.3.3 Zero-Shot Learning

We evaluate UDify for zero-shot learning, i.e., evaluating predicted annotations for languages that have not been trained on UDify. Table 4.4 displays a subset of test set evaluations of treebanks that do not have a training set. The zero-shot results indicate that fine-tuning on BERT can result in reasonably high scores on these languages.

TREEBANK		UPOS	FEATS	LEM	UAS	LAS
Breton KEB	**br_keb**	63.67	46.75	53.15	63.97	40.19
Tagalog TRG	**tl_trg**	61.64	35.27	75.00	64.73	39.38
Faroese OFT	fo_oft	77.86	35.71	53.82	69.28	61.03
Naija NSC	pcm_nsc	56.59	52.75	97.52	47.13	33.43
Sanskrit UFAL	sa_ufal	40.21	18.45	37.60	41.73	19.80

Table 4.4: Test set results for zero-shot learning, i.e., no UD training annotations available. Languages that are pretrained with BERT are bolded.

It can be seen that a combination of BERT pretraining and multilingual learning can improve predictions for Breton and Tagalog, which implies that the network has

learned representations of syntax that cross lingual boundaries. Furthermore, despite the fact that neither BERT nor UDify have directly observed Faroese, Naija, or Sanskrit, we see unusually high performance in these languages. This can be partially attributed to each language closely resembling another: Faroese is very close to Icelandic, Naija (Nigerian Pidgin) is a variant of English, and Sanskrit is an ancient Indian language related to Greek, Latin, and Hindi.

4.3.4 Probing for Syntax

Hewitt and Manning [2019] introduce a structural probe for identifying dependency trees in contextualized word embeddings. This probe evaluates whether syntax trees (i.e., unlabeled undirected dependency trees) are embedded as a linear transformation of the network's contextual word embeddings. The probe trains a weight matrix on the final layer of contextual embeddings produced by BERT, identifying a linear transformation where squared L2 distance between embeddings encodes the distance between words in the parse tree, and squared L2 norm encodes depth in the parse tree. Once trained, the probe can use the resulting probabilities in the matrix to extract the most probable undirected spanning tree across the sentence. Despite the fact BERT and the probe are trained without any annotated examples of real dependency trees, the probe reveals that these global properties of the embeddings result in spanning trees that are very close to human annotated dependency trees.

We train the structural probe on unmodified and fine-tuned BERT using the default hyperparameters of Hewitt and Manning [2019] to evaluate whether the representations affected by fine-tuning BERT on dependency trees would more closely match the structure of these trees.

Table 4.5 compares the unlabeled undirected attachment scores (UUAS) of dependency trees produced using the structural probe on both the unmodified multilingual cased BERT model and the extracted BERT model fine-tuned on the English EWT treebank.

TREEBANK	MODEL	UUAS
English EWT (en_ewt)	BERT	65.48
	BERT+finetune en_ewt	**79.67**

Table 4.5: UUAS test scores calculated on the predictions produced by the syntactic structural probe using the English EWT treebank, on the unmodified multilingual cased BERT model and the same BERT model fine-tuned on the treebank.

We see that even without any supervised training, BERT encodes its syntax in the embedding's distance and norm surprisingly close to human-annotated dependencies. The results show that fine-tuning BERT on Universal Dependencies significantly boosts UUAS scores when compared to the gold dependency trees, an error reduction of 41%. This indicates that the self-attention weights have learned a linearly-transformable representation more closely resembling annotated dependency trees defined by linguists.

To provide a more clear illustration of the probe and the differences between before and after fine-tuning, we provide several examples of trees produced by the probe in

Figure 4.3 and Figure 4.4. Sentences are repeated next to each other, showing trees extracted by the syntactic probe by BERT (before fine-tuning) in red and UDify (after fine-tuning) in blue. The human annotated dependency trees are provided in black.

Figure 4.3 shows examples where BERT gets close to producing spanning trees in the attention weights that are the same as the dependency trees, but not quite. In these instances, the fine-tuned UDify model gets them exactly right. This shows that with some supervised fine-tuning, BERT will start to encode a property of the distances of vectors in the embedding vector space that makes it nearly trivial to learn and apply a linear operation to produce the correct dependency trees.

Figure 4.4 shows probed trees like in Figure 4.3, but with some examples where UDify is not completely correct. Not only are the probed trees produced by UDify closer to the gold annotations, we see that UDify is much more aware of constituent boundaries. The trees show instances where the representations do not cross dependencies across punctuation like commas, and are less likely to cross between adjectives and determiners. The result is that UDify is able to chunk sentences of English, demonstrating a capability more aware of phrasal boundaries.

This also demonstrates why BERT can produce high-quality syntax so early on in fine-tuning with just a few epochs, as shown in Devlin et al. [2018]. The syntactic probe shows that even with just unsupervised pretraining, a global structural property of the vector space of the BERT weights already produces a decent representation of the dependency tree in the L2 distance/norm. Following this, it should be no surprise that further fine-tuning with a non-linear graph-based dependency decoder on human annotations would result in even higher quality dependency trees embedded in the vector space.

4.3.5 Attention Visualization

We perform a high-level visual analysis of the BERT attention weights to see if they have changed on any discernible level. Our observations reveal something notable: the attention weights tend to be more sparse, and are more often sensitive to constituent boundaries like clauses and prepositions. Figures 4.5, 4.6 4.7, and 4.8 illustrate this point, showing the attention weights of a particular attention head on an example sentence. We find similar behavior in 9 additional attention heads for this particular sentence.

In Figure 4.5, we see that some of the attention structure remains after fine-tuning. Previously, the attention head was mostly sensitive to previous words and punctuation. But after fine-tuning, it demonstrates more fine-grained attention towards immediate wordpieces, prepositions, articles, and adjectives. We found similar evidence in other attention heads, which implies that fine-tuning on UD produces attention that more closely resembles localized dependencies within constituents. We also find that BERT base heavily preferred to attend to punctuation, while UDify BERT does to a much lesser degree. We can see similar behavior in Figures 4.6 and 4.7.

Not all heads become sparse with respect to constituents, however. We see in Figure 4.8 that there exists an attention head that has become most sensitive to the main subjects in the sentence, in this case, the pronoun “he.” Other attention heads exhibit similar behavior, specializing in certain roles more often than taking the combination of outputs over all words. While we cannot say anything definitive on how this results in improved performance, we can say that these attention structures more closely re-

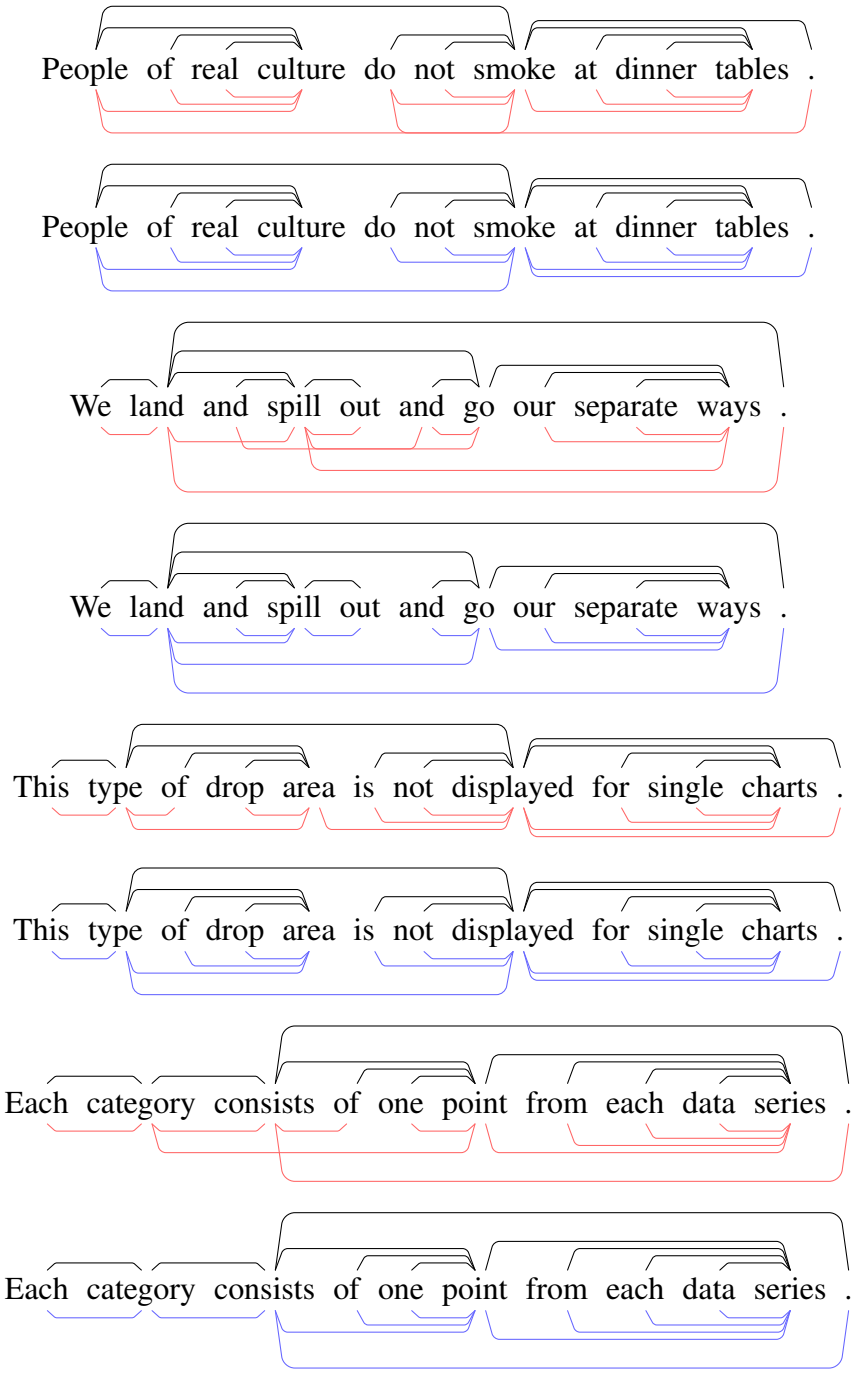

Figure 4.3: Examples of minimum spanning trees produced by the syntactic probe are shown below each sentence, evaluated on BERT (red) and on UDify (blue). Human annotated dependency trees are shown above each sentence in black.

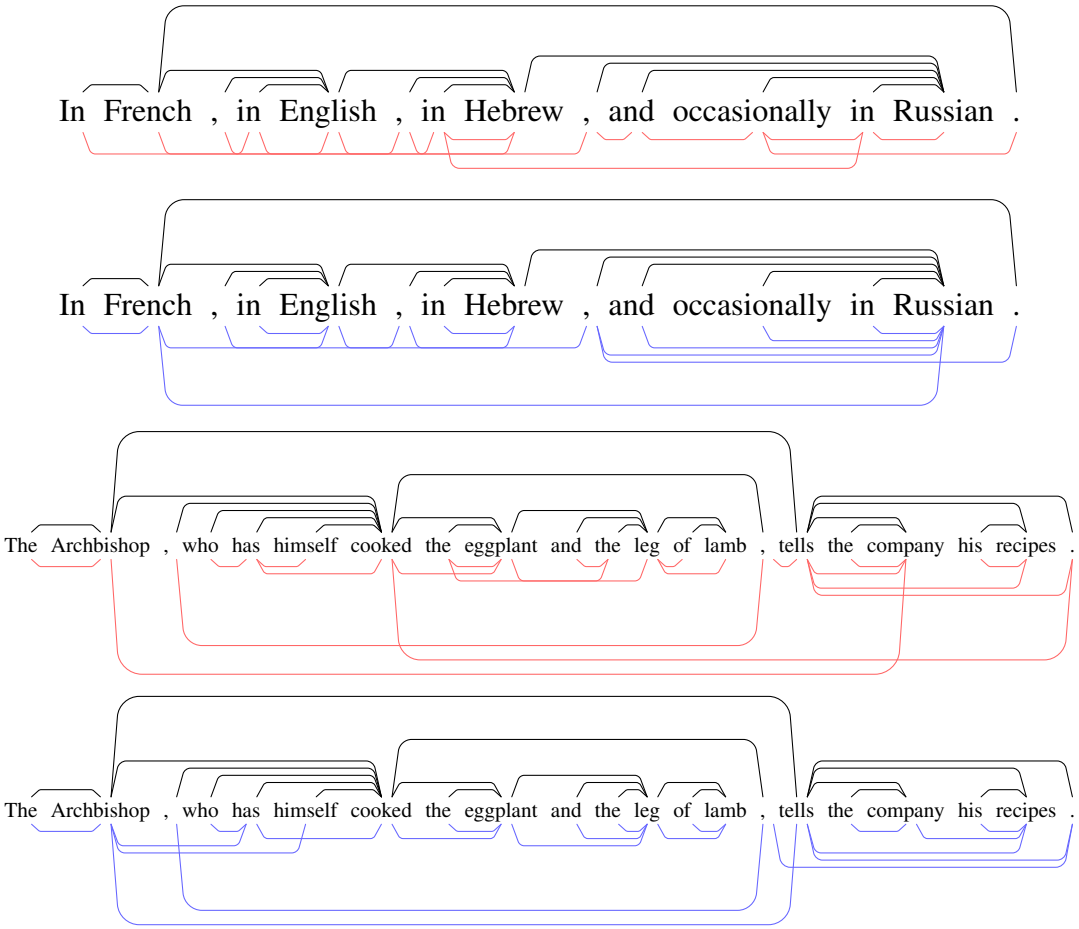

Figure 4.4: Examples of minimum spanning trees produced by the syntactic probe are shown below each sentence, evaluated on BERT (red) and on UDify (blue). Human annotated dependency trees are shown above each sentence in black.

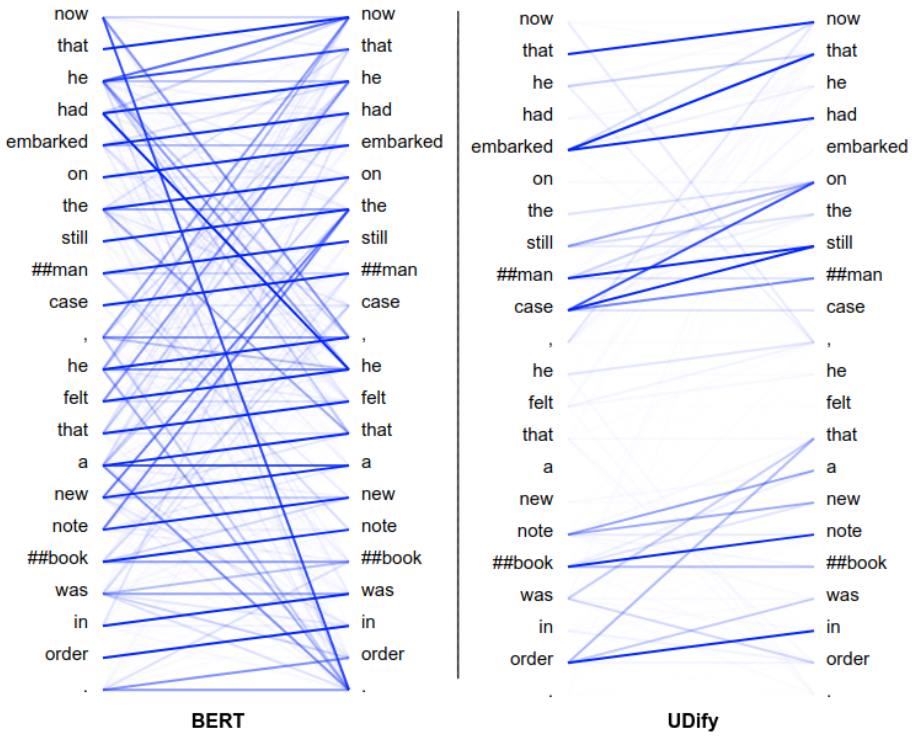

Figure 4.5: Visualization of BERT attention head 4 at layer 11, comparing the attended words on an English sentence between BERT base and UDify BERT. The right column indicates the attended words (keys) with respect to the words in the left column (queries). Darker lines indicate stronger attention weights.

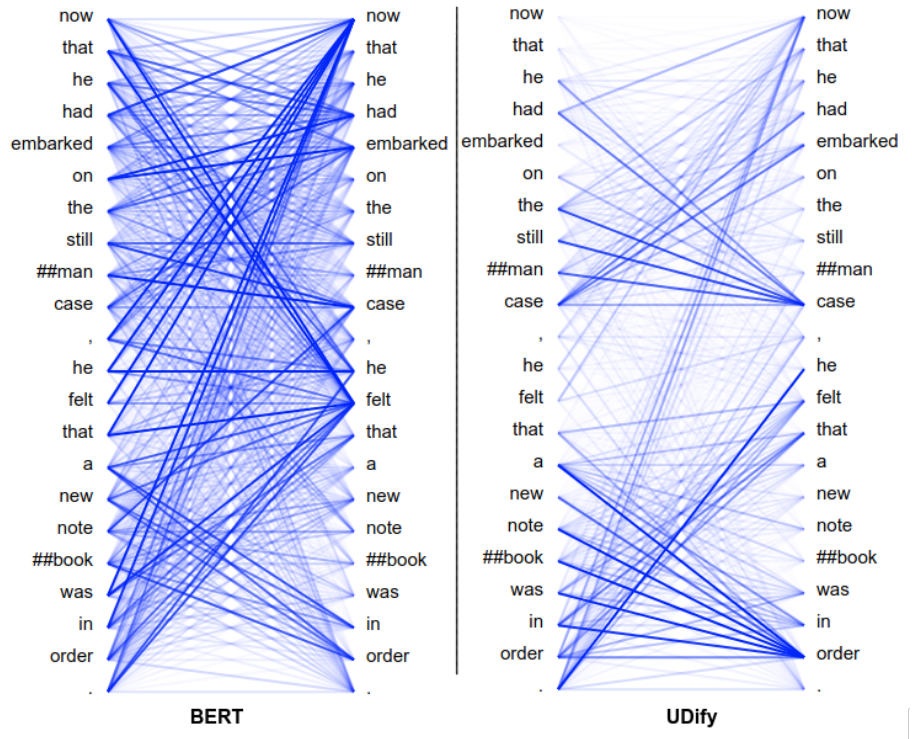

Figure 4.6: Visualization of BERT attention head 3 at layer 7, comparing the attended words on an English sentence between BERT base and UDify BERT. The right column indicates the attended words (keys) with respect to the words in the left column (queries). Darker lines indicate stronger attention weights.

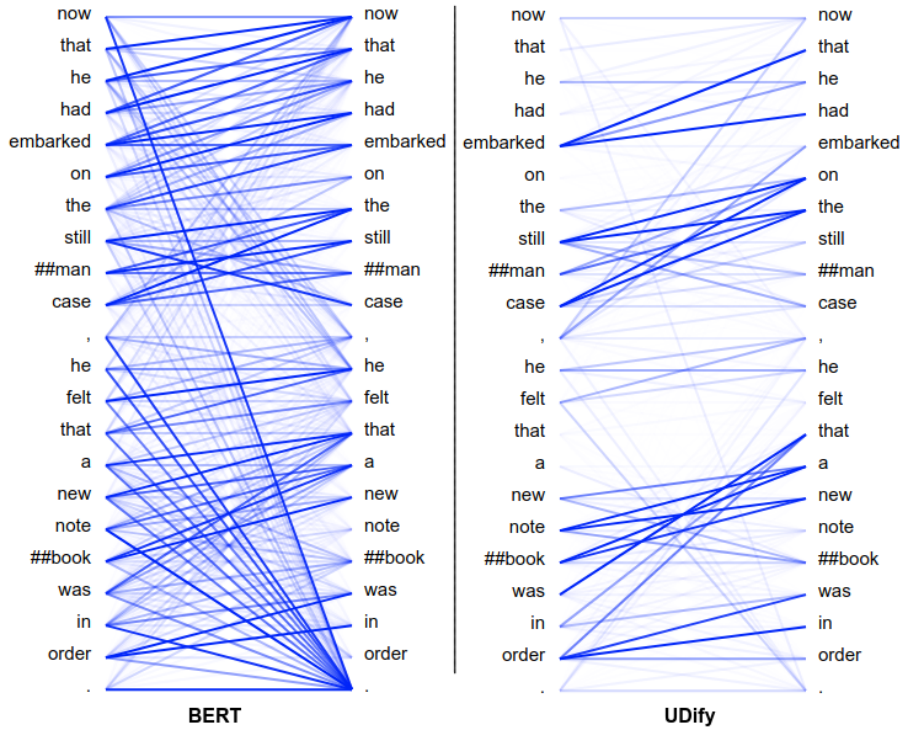

Figure 4.7: Visualization of BERT attention head 8 at layer 10, comparing the attended words on an English sentence between BERT base and UDify BERT. The right column indicates the attended words (keys) with respect to the words in the left column (queries). Darker lines indicate stronger attention weights.

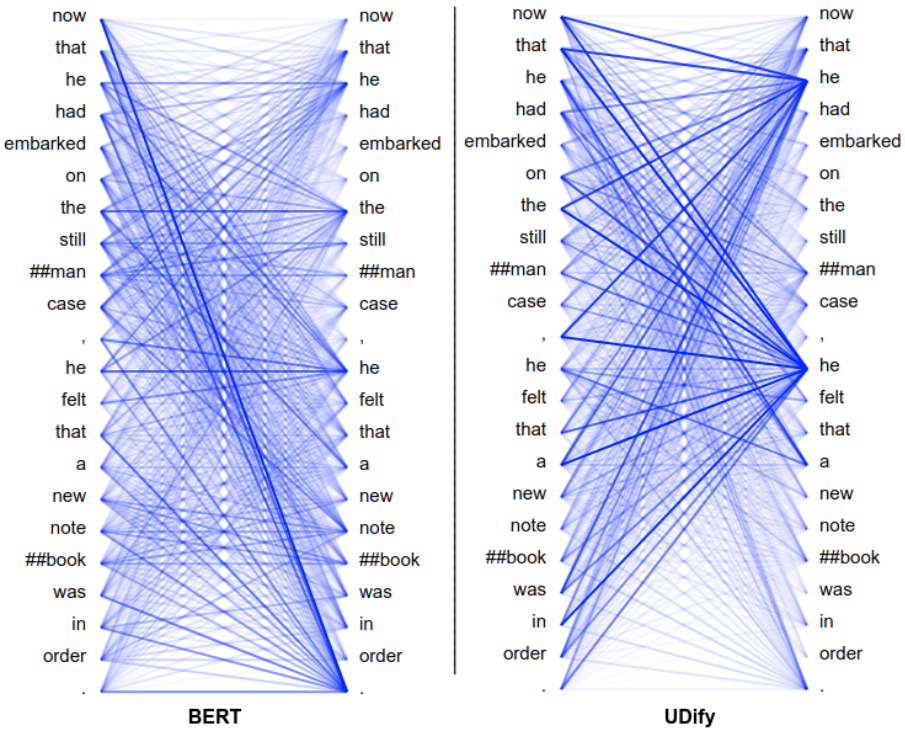

Figure 4.8: Visualization of BERT attention head 12 at layer 10, comparing the attended words on an English sentence between BERT base and UDify BERT. The right column indicates the attended words (keys) with respect to the words in the left column (queries). Darker lines indicate stronger attention weights.

semble the types of chunking of English phrases that linguists typically employ, and that they appear to capture more nuanced structures of the input sentence.

4.4 Factors that Enable BERT to Excel at Dependency Parsing and Multilinguality

Goldberg [2019] assesses the syntactic capabilities of BERT and concludes that BERT is remarkably capable of processing syntactic tasks despite not being trained on any supervised data. Conducting similar experiments, Vig [2019] and Sileo [2019] visualize the attention heads within each BERT layer, showing a number of distinct attention patterns, including attending to previous/next words, related words, punctuation, verbs/nouns, and coreference dependencies. This neat delegation of certain low-level information processing tasks to attention heads hints at why BERT might excel at processing syntax.

We see that from the analysis on BERT fine-tuned with syntax using the syntactic probe and attention visualization, BERT produces a representation that keeps constituents close in vector space, and improves this representation to more closely resemble human annotated dependency trees when fine-tuned on UD. Furthermore, Ahmad et al. [2018] claim that self-attention networks can be more robust than recurrent networks to the change of word order, observing that self-attention networks capture less word order information in their architecture, which is what allows them to generally perform better at cross-lingual parsing. Ahmad et al. [2018] also show results consistent with their claim that self-attention networks perform well on distant languages, while recurrent networks generally perform poorly with languages that are not structured similarly with respect to word order. These findings also suggest that this word order property inherent in recurrent networks provide a bottleneck that prevents the network from generalizing across word order past a certain point, while large enough self-attention networks can scale away this bottleneck with very little degradation in performance. For instance, Devlin et al. [2018] show that the fine-tuning on the multilingual BERT model only results in about 3% degradation in performance compared to an equivalent model pretrained only on English.

From the evidence above, we can see that the combination of strong regularization paired with the ability to capture long dependencies with self-attention and contextual pretraining on an enormous corpus of raw text are large contributors that enable strong multilingual modeling with respect to dependency parsing. The dependencies seen in the output dependency trees are highly correlated with the implicit dependencies learned by the self-attention, showing that self-attention is remarkably capable of modeling syntax, indicating a more interpretable representation modeling word-level dependencies than plain recurrent networks. The introduction of multilingual data also shows that these attention heads provide a surprising amount of capacity that does not degrade the performance significantly when compared to monolingual training, hinting that the BERT model can be compressed significantly without compromising on evaluation performance.

Our results show that modeling language-specific properties are not strictly necessary to achieve high-performing cross-lingual representations for dependency parsing, though we caution that the model can also likely be improved by these techniques.

4.5 Full Results

We show in Tables 4.6, 4.7, 4.8, and 4.9 UDify scores evaluated on all 124 treebanks with the official CoNLL 2018 Shared Task evaluation script. For comparison, we also include the full test evaluation of UDPipe Future on the subset of 89 treebanks with a training set. We also include additional metrics provided by the CoNLL evaluation script, whose calculation is explained by the CoNLL 2018 website[2].

One aspect to note is that UDify severely underperforms the baseline on a few low-resource languages, e.g., cop_scriptorum in Table 4.6. We surmise that this is due to using mixed batches on an unbalanced training set, which skews the model towards predicting larger treebanks more accurately. However, we find that fine-tuning on the treebank individually with BERT weights saved from UDify eliminates most of these gaps in performance.

Due to excessive resource requirements, we were unable to perform a more extensive ablation into all of the architectural choices we made, let alone across all languages. UDify required approximately one month to train on the whole of UD before the model stopped improving, using a single NVIDIA GTX 1080 Ti GPU. We would also like to more directly compare to other models submitted to the CoNLL 2018 shared task which provides predicted segmentations of each model, but we argue that evaluating UDify on these segmentations would not be a fair comparison, as UDify was never trained with those segmentations in mind. Therefore, we only include the results of training UDPipe Future on the gold segmentation to provide a fair representation of model performance.

[2] https://universaldependencies.org/conll18/evaluation.html

Treebank		Model	UPOS	UFeats	Lemmas	UAS	LAS	CLAS	MLAS	BLEX
Afrikaans AfriBooms	af_afribooms	UDPipe	**98.25**	**97.66**	**97.46**	**89.38**	**86.58**	**81.44**	**77.66**	**77.82**
		UDify	97.48	96.63	95.23	86.97	83.48	77.42	70.57	70.93
Akkadian PISANDUB	akk_pisandub	UDify	**19.92**	**99.51**	**2.32**	**27.65**	**4.54**	**3.27**	**1.04**	**0.30**
Amharic ATT	am_att	UDify	**15.25**	**43.95**	**58.04**	**17.38**	**3.49**	**4.88**	**0.23**	**2.53**
Ancient Greek PROIEL	grc_proiel	UDPipe	**97.86**	**92.44**	**93.51**	**85.93**	**82.11**	**77.70**	**67.16**	**71.22**
		UDify	91.20	82.29	76.16	78.91	72.66	66.07	50.79	47.27
Ancient Greek Perseus	grc_perseus	UDPipe	**93.27**	**91.39**	**85.02**	**78.85**	**73.54**	**67.60**	**53.87**	**53.19**
		UDify	85.67	81.67	70.51	70.51	62.64	55.60	39.15	35.05
Arabic PADT	ar_padt	UDPipe	**96.83**	**94.11**	**95.28**	87.54	**82.94**	**79.77**	**73.92**	**75.87**
		UDify	96.58	91.77	73.55	**87.72**	82.88	79.47	70.52	50.26
Arabic PUD	ar_pud	UDify	**79.98**	**40.32**	**0.00**	**76.17**	**67.07**	**65.10**	**10.67**	**0.00**
Armenian ArmTDP	hy_armtdp	UDPipe	93.49	**82.85**	**92.86**	78.62	71.27	65.77	**48.11**	**60.11**
		UDify	**94.42**	76.90	85.63	**85.63**	**78.61**	**73.72**	46.80	59.14
Bambara CRB	bm_crb	UDify	**30.86**	**57.96**	**20.42**	**30.28**	**8.60**	**6.56**	**1.04**	**0.76**
Basque BDT	eu_bdt	UDPipe	**96.11**	**92.48**	**96.29**	**86.11**	**82.86**	**81.79**	**72.33**	**78.54**
		UDify	95.45	86.80	90.53	84.94	80.97	79.52	63.60	71.56
Belarusian HSE	be_hse	UDPipe	93.63	73.30	**87.34**	78.58	72.72	69.14	46.20	58.28
		UDify	**97.54**	**89.36**	85.46	**91.82**	**87.19**	**85.05**	**71.54**	**68.66**
Breton KEB	br_keb	UDify	**62.78**	**47.12**	**51.31**	**63.52**	**39.84**	**35.14**	**4.64**	**16.34**
Bulgarian BTB	bg_btb	UDPipe	**98.98**	**97.82**	**97.94**	93.38	90.35	87.01	**83.63**	**84.42**
		UDify	98.89	96.18	93.49	**95.54**	**92.40**	**89.59**	83.43	80.44
Buryat BDT	bxr_bdt	UDPipe	40.34	32.40	58.17	32.60	18.83	12.36	1.26	**6.49**
		UDify	**61.73**	**47.45**	**61.03**	**48.43**	**26.28**	**20.61**	**5.51**	11.68
Cantonese HK	yue_hk	UDify	**67.11**	**91.01**	**96.01**	**46.82**	**32.01**	**33.35**	**14.29**	**31.26**
Catalan AnCora	ca_ancora	UDPipe	98.88	**98.37**	**99.07**	93.22	91.06	87.18	84.48	86.18
		UDify	**98.89**	98.34	98.14	**94.25**	**92.33**	**89.27**	**86.21**	**86.61**
Chinese CFL	zh_cfl	UDify	**83.75**	**82.72**	**98.75**	**62.46**	**42.48**	**43.46**	**21.07**	**42.22**
Chinese GSD	zh_gsd	UDPipe	94.88	99.22	**99.99**	84.64	80.50	76.79	71.04	76.78
		UDify	**95.35**	**99.35**	99.97	**87.93**	**83.75**	**80.33**	**74.36**	**80.28**
Chinese HK	zh_hk	UDify	**82.86**	**86.47**	**100.00**	**65.53**	**49.32**	**47.84**	**22.85**	**47.84**
Chinese PUD	zh_pud	UDify	**92.68**	**98.40**	**100.00**	**79.08**	**56.51**	**55.22**	**40.92**	**55.22**
Coptic Scriptorium	cop_scriptorium	UDPipe	**94.70**	**96.35**	**95.49**	**85.58**	**80.97**	**72.24**	**64.45**	**68.48**
		UDify	27.17	52.85	55.71	27.58	10.82	6.50	0.19	1.44
Croatian SET	hr_set	UDPipe	**98.13**	**92.25**	**97.27**	91.10	86.78	84.11	**73.61**	81.19
		UDify	98.02	89.67	95.34	**94.08**	**89.79**	**87.70**	72.72	**82.00**
Czech CAC	cs_cac	UDPipe	**99.37**	**96.34**	**98.57**	92.99	90.71	88.84	84.30	87.18
		UDify	99.14	95.42	98.32	**94.33**	**92.41**	**91.03**	**84.68**	**89.21**
Czech CLTT	cs_cltt	UDPipe	98.88	91.59	98.25	86.90	84.03	80.55	71.63	79.20
		UDify	**99.17**	**93.66**	**98.86**	**91.69**	**89.96**	**87.59**	**79.50**	**86.79**
Czech FicTree	cs_fictree	UDPipe	**98.55**	**95.87**	**98.63**	92.91	89.75	86.97	**81.04**	85.49
		UDify	98.34	91.82	98.13	**95.19**	**92.77**	**90.99**	77.77	**88.39**
Czech PDT	cs_pdt	UDPipe	**99.18**	**97.23**	**99.02**	93.33	91.31	89.64	86.15	88.60
		UDify	**99.18**	96.69	98.52	**94.73**	**92.88**	**91.64**	**87.13**	**89.95**
Czech PUD	cs_pud	UDify	**97.93**	**93.98**	**96.94**	**92.59**	**87.95**	**84.85**	**77.39**	**82.81**

Table 4.6: The full test results of UDify on 124 treebanks (part 1 of 5).

Treebank		Model	UPOS	UFeats	Lemmas	UAS	LAS	CLAS	MLAS	BLEX
Danish DDT	da_ddt	UDPipe	**97.78**	**97.33**	**97.52**	86.88	84.31	81.20	**76.29**	**78.51**
		UDify	97.50	95.41	94.60	**87.76**	**84.50**	**81.60**	73.76	75.15
Dutch Alpino	nl_alpino	UDPipe	96.83	96.33	**97.09**	91.37	88.38	83.51	77.28	79.82
		UDify	**97.67**	**97.66**	95.44	**94.23**	**91.21**	**87.32**	**82.81**	**80.76**
Dutch LassySmall	nl_lassysmall	UDPipe	96.50	96.42	**97.41**	90.20	86.39	81.88	77.19	78.83
		UDify	**96.70**	**96.57**	95.10	**94.34**	**91.22**	**88.03**	**82.06**	**81.40**
English EWT	en_ewt	UDPipe	**96.29**	**97.10**	**98.25**	89.63	86.97	84.02	79.00	82.36
		UDify	96.21	96.02	97.28	**90.96**	**88.50**	**86.25**	**79.80**	**83.39**
English GUM	en_gum	UDPipe	**96.02**	**96.82**	**96.85**	87.27	84.12	78.55	**73.51**	**74.68**
		UDify	95.44	94.12	93.15	**89.14**	**85.73**	**83.03**	72.55	74.30
English LinES	en_lines	UDPipe	**96.91**	**96.31**	**96.45**	84.15	79.71	77.44	**71.38**	73.22
		UDify	95.31	91.34	94.50	**87.33**	**83.71**	**82.95**	68.62	**76.23**
English PUD	en_pud	UDify	**96.18**	**93.50**	**94.20**	**91.52**	**88.66**	**87.83**	**75.61**	**80.57**
English ParTUT	en_partut	UDPipe	96.10	**95.51**	**97.74**	90.29	87.27	82.58	**76.44**	80.33
		UDify	**96.16**	92.61	96.45	**92.84**	**90.14**	**86.28**	74.59	**82.01**
Erzya JR	myv_jr	UDify	**46.66**	**31.82**	**45.73**	**31.90**	**16.38**	**10.83**	**0.58**	**2.83**
Estonian EDT	et_edt	UDPipe	**97.64**	**96.23**	**95.30**	88.00	85.18	83.62	78.72	**78.51**
		UDify	97.44	95.13	86.56	**89.53**	**86.67**	**85.17**	**79.20**	69.31
Faroese OFT	fo_oft	UDify	**77.46**	**35.20**	**51.09**	**67.24**	**59.26**	**51.17**	**2.39**	**21.92**
Finnish FTB	fi_ftb	UDPipe	**96.65**	**96.62**	**95.49**	**90.68**	**87.89**	**85.11**	**80.58**	**81.18**
		UDify	93.80	90.38	88.80	86.37	81.40	81.01	68.16	70.15
Finnish PUD	fi_pud	UDify	**96.48**	**93.84**	**84.64**	**89.76**	**86.58**	**86.64**	**77.83**	**69.12**
Finnish TDT	fi_tdt	UDPipe	**97.45**	**95.43**	**91.45**	**89.88**	**87.46**	**85.87**	**80.43**	**76.64**
		UDify	94.43	90.48	82.89	86.42	82.03	82.62	70.89	63.66
French GSD	fr_gsd	UDPipe	97.63	**97.13**	**98.35**	90.65	88.06	84.35	79.76	82.39
		UDify	**97.83**	96.17	97.34	**93.60**	**91.45**	**88.54**	**81.61**	**84.51**
French PUD	fr_pud	UDify	**91.67**	**59.65**	**100.00**	**88.36**	**82.76**	**81.74**	**25.24**	**81.74**
French ParTUT	fr_partut	UDPipe	**96.93**	**94.43**	**95.70**	**92.17**	**89.63**	**84.62**	**75.22**	**78.07**
		UDify	96.12	88.36	93.97	90.55	88.06	83.19	63.03	74.03
French Sequoia	fr_sequoia	UDPipe	**98.79**	**98.09**	**98.57**	92.37	**90.73**	**87.55**	**84.51**	**85.93**
		UDify	97.89	88.97	97.15	**92.53**	90.05	86.67	67.98	82.52
French Spoken	fr_spoken	UDPipe	95.91	100.00	**96.92**	82.90	77.53	71.82	68.24	69.47
		UDify	**96.23**	**98.67**	96.59	**85.24**	**80.01**	**75.40**	**69.74**	**72.77**
Galician CTG	gl_ctg	UDPipe	**97.84**	**99.83**	**98.58**	**86.44**	**83.82**	**78.58**	**72.46**	**77.21**
		UDify	96.51	97.10	97.08	84.75	80.89	74.62	65.86	72.17
Galician TreeGal	gl_treegal	UDPipe	**95.82**	**93.96**	**97.06**	82.72	**77.69**	71.69	**63.73**	**68.89**
		UDify	94.59	80.67	94.93	**84.08**	76.77	**73.06**	49.76	66.99
German GSD	de_gsd	UDPipe	94.48	**90.68**	**96.80**	85.53	81.07	76.26	58.82	72.13
		UDify	**94.55**	90.43	94.42	**87.81**	**83.59**	**80.03**	**61.27**	**72.48**
German PUD	de_pud	UDify	**89.49**	**30.66**	**94.77**	**89.86**	**84.46**	**80.50**	**2.10**	**72.95**
Gothic PROIEL	got_proiel	UDPipe	**96.61**	**90.73**	**94.75**	85.28	**79.60**	**76.92**	**66.70**	**72.93**
		UDify	95.55	85.97	80.57	**85.61**	79.37	76.26	63.09	58.65
Greek GDT	el_gdt	UDPipe	**97.98**	**94.96**	**95.82**	92.10	89.79	85.71	**78.60**	**79.72**
		UDify	97.72	93.29	89.43	**94.33**	**92.15**	**88.67**	77.89	71.83
Hebrew HTB	he_htb	UDPipe	**97.02**	**95.87**	**97.12**	89.70	86.86	81.45	**75.52**	**78.14**
		UDify	96.94	93.41	94.15	**91.63**	**88.11**	**83.04**	72.55	74.87

Table 4.7: The full test results of UDify on 124 treebanks (part 2 of 5).

TREEBANK		MODEL	UPOS	UFEATS	LEMMAS	UAS	LAS	CLAS	MLAS	BLEX
Hindi HDTB	hi_hdtb	UDPipe	**97.52**	**94.15**	**98.67**	94.85	**91.83**	**88.21**	**78.49**	**86.83**
		UDify	97.12	92.59	98.23	**95.13**	91.46	87.80	75.54	86.10
Hindi PUD	hi_pud	UDify	**87.54**	**22.81**	**100.00**	**71.64**	**58.42**	**53.03**	**3.32**	**53.03**
Hungarian Szeged	hu_szeged	UDPipe	95.76	**91.75**	**95.05**	84.04	79.73	78.65	**67.63**	**73.63**
		UDify	**96.36**	86.16	90.19	**89.68**	**84.88**	**83.93**	64.27	72.21
Indonesian GSD	id_gsd	UDPipe	**93.69**	**95.58**	**99.64**	85.31	78.99	76.76	**67.74**	**76.38**
		UDify	93.36	93.32	98.37	**86.45**	**80.10**	**78.05**	66.93	76.31
Indonesian PUD	id_pud	UDify	**76.10**	**44.23**	**100.00**	**77.47**	**56.90**	**54.88**	**7.41**	**54.88**
Irish IDT	ga_idt	UDPipe	**92.72**	**82.43**	**90.48**	**80.39**	**72.34**	**63.48**	**46.49**	**55.32**
		UDify	90.49	71.84	81.27	80.05	69.28	60.02	34.39	43.07
Italian ISDT	it_isdt	UDPipe	98.39	**98.11**	**98.66**	93.49	91.54	87.34	84.28	85.49
		UDify	**98.51**	98.01	97.72	**95.54**	**93.69**	**90.40**	**86.54**	**86.70**
Italian PUD	it_pud	UDify	**94.73**	**58.16**	**96.08**	**94.18**	**91.76**	**90.05**	**25.55**	**83.74**
Italian ParTUT	it_partut	UDPipe	**98.38**	97.77	**98.16**	92.64	90.47	85.05	81.87	82.99
		UDify	98.21	**98.38**	97.55	**95.96**	**93.68**	**89.83**	**86.83**	**86.44**
Italian PoSTWITA	it_postwita	UDPipe	**96.61**	**96.90**	**97.00**	86.03	81.78	77.05	**72.88**	**74.33**
		UDify	96.14	95.56	93.60	**87.20**	**82.77**	**78.27**	72.41	71.67
Japanese GSD	ja_gsd	UDPipe	**98.13**	**99.98**	**99.52**	**95.06**	**93.73**	**88.35**	**86.37**	**88.04**
		UDify	97.08	99.97	98.80	94.37	92.08	86.19	82.99	85.12
Japanese Modern	ja_modern	UDify	**74.94**	**96.14**	**79.70**	**74.99**	**55.62**	**42.67**	**30.89**	**35.47**
Japanese PUD	ja_pud	UDify	**97.89**	**99.98**	**99.31**	**94.89**	**93.62**	**87.92**	**84.86**	**87.15**
Kazakh KTB	kk_ktb	UDPipe	55.84	40.40	63.96	53.30	33.38	27.06	**4.82**	15.10
		UDify	**85.59**	**65.49**	**77.18**	**74.77**	**63.66**	**61.84**	34.23	**45.51**
Komi Zyrian IKDP	kpv_ikdp	UDify	**59.92**	**39.32**	**57.56**	**36.01**	**22.12**	**17.45**	**1.54**	**6.80**
Komi Zyrian Lattice	kpv_lattice	UDify	**38.57**	**29.45**	**55.33**	**28.85**	**12.99**	**10.79**	**0.72**	**3.28**
Korean GSD	ko_gsd	UDPipe	**96.29**	**99.77**	**93.40**	**87.70**	**84.24**	**82.05**	**79.74**	**76.35**
		UDify	90.56	99.63	82.84	82.74	74.26	71.72	65.94	57.58
Korean Kaist	ko_kaist	UDPipe	**95.59**	100.00	**94.30**	**88.42**	**86.48**	**84.12**	**80.72**	**79.22**
		UDify	94.67	**99.98**	85.89	87.57	84.52	82.05	78.27	68.99
Korean PUD	ko_pud	UDify	**64.43**	**60.47**	**70.47**	**63.57**	**46.89**	**45.29**	**16.26**	**30.94**
Kurmanji MG	kmr_mg	UDPipe	53.36	**41.54**	**69.58**	**45.23**	**34.32**	**29.41**	**2.74**	19.39
		UDify	**60.23**	37.78	58.08	35.86	20.40	14.75	1.42	**7.28**
Latin ITTB	la_ittb	UDPipe	98.34	**96.97**	**98.99**	91.06	88.80	86.40	**82.35**	85.71
		UDify	**98.48**	95.81	98.08	**92.43**	**90.12**	**87.93**	82.24	**85.97**
Latin PROIEL	la_proiel	UDPipe	**97.01**	**91.53**	**96.32**	83.34	78.66	76.20	**67.40**	**73.65**
		UDify	96.79	89.49	91.79	**84.85**	**80.52**	**77.96**	67.18	71.00
Latin Perseus	la_perseus	UDPipe	88.40	79.10	**81.45**	71.20	61.28	56.32	41.58	45.09
		UDify	**90.96**	**82.09**	81.08	**78.33**	**69.60**	**65.95**	**50.26**	**51.33**
Latvian LVTB	lv_lvtb	UDPipe	**96.11**	**93.01**	**95.46**	87.20	83.35	80.90	**71.92**	**76.64**
		UDify	96.02	89.78	91.00	**89.33**	**85.09**	**82.34**	69.51	72.58
Lithuanian HSE	lt_hse	UDPipe	81.70	60.47	**76.89**	51.98	42.17	38.93	18.17	28.70
		UDify	**90.47**	**70.00**	67.17	**79.06**	**69.34**	**66.00**	**36.21**	**36.35**
Maltese MUDT	mt_mudt	UDPipe	**95.99**	100.00	**100.00**	**84.65**	**79.71**	**71.49**	**66.75**	**71.49**
		UDify	91.98	**99.89**	**100.00**	83.07	75.56	65.08	58.14	65.08
Marathi UFAL	mr_ufal	UDPipe	80.10	**67.23**	**81.31**	70.63	61.41	57.44	**29.34**	**45.87**
		UDify	**88.59**	59.22	72.82	**79.37**	**67.72**	**60.13**	21.71	39.25

Table 4.8: The full test results of UDify on 124 treebanks (part 3 of 5).

TREEBANK		MODEL	UPOS	UFEATS	LEMMAS	UAS	LAS	CLAS	MLAS	BLEX
Naija NSC	pcm_nsc	UDify	**55.44**	**51.32**	**97.03**	**45.75**	**32.16**	**31.62**	**4.73**	**29.33**
North Sami Giella	sme_giella	UDPipe	**92.54**	**90.03**	**88.31**	**78.30**	**73.49**	**70.94**	**62.40**	**61.45**
		UDify	90.21	83.55	71.50	74.30	67.13	64.41	51.20	40.63
Norwegian Bokmaal	no_bokmaal	UDPipe	**98.31**	**97.14**	**98.64**	92.39	90.49	88.18	84.06	86.53
		UDify	98.18	96.36	97.33	**93.97**	**92.18**	**90.40**	**85.02**	**87.13**
Norwegian Nynorsk	no_nynorsk	UDPipe	**98.14**	**97.02**	**98.18**	92.09	90.01	87.68	82.97	85.47
		UDify	**98.14**	96.55	97.18	**94.34**	**92.37**	**90.39**	**85.01**	**86.71**
Norwegian NynorskLIA	no_nynorsklia	UDPipe	89.59	86.13	93.93	68.08	60.07	54.89	44.47	50.98
		UDify	**95.01**	**93.36**	**96.13**	**75.40**	**69.60**	**65.33**	**56.90**	**62.27**
Old Church Slavonic PROIEL	cu_proiel	UDPipe	**96.91**	**90.66**	**93.11**	**89.66**	**85.04**	**83.41**	**73.63**	**77.81**
		UDify	84.23	71.30	65.70	76.71	66.67	64.10	46.25	43.88
Old French SRCMF	fro_srcmf	UDPipe	**96.09**	**97.81**	**100.00**	**91.74**	**86.83**	**83.85**	**79.91**	**83.85**
		UDify	95.73	96.98	**100.00**	**91.74**	86.65	83.49	78.85	83.49
Persian Seraji	fa_seraji	UDPipe	**97.75**	**97.78**	**97.44**	**90.05**	**86.66**	**83.26**	**81.23**	**80.93**
		UDify	96.22	94.73	92.55	89.59	85.84	81.98	76.65	74.74
Polish LFG	pl_lfg	UDPipe	**98.80**	**95.49**	**97.54**	96.58	**94.76**	93.01	**87.04**	**90.26**
		UDify	**98.80**	87.71	94.04	**96.67**	94.58	**93.03**	76.50	85.15
Polish SZ	pl_sz	UDPipe	98.34	**93.04**	**97.16**	93.39	**91.24**	**89.39**	**81.06**	**85.99**
		UDify	**98.36**	67.11	93.92	**93.67**	89.20	87.31	48.47	80.24
Portuguese Bosque	pt_bosque	UDPipe	97.07	**96.40**	**98.46**	91.36	**89.04**	**85.19**	**76.67**	**83.06**
		UDify	**97.10**	89.70	91.60	**91.37**	87.84	84.13	69.09	78.64
Portuguese GSD	pt_gsd	UDPipe	**98.31**	**99.92**	**99.30**	93.01	91.63	87.67	**85.96**	86.94
		UDify	98.04	95.75	98.95	**94.22**	**92.54**	**89.37**	82.32	**87.90**
Portuguese PUD	pt_pud	UDify	**90.14**	**51.16**	**99.79**	**87.02**	**80.17**	**74.10**	**17.51**	**74.10**
Romanian Nonstandard	ro_nonstandard	UDPipe	96.68	**90.88**	**94.78**	89.12	84.20	78.91	**65.93**	**73.44**
		UDify	**96.83**	88.89	89.33	**90.36**	**85.26**	**80.41**	64.68	68.11
Romanian RRT	ro_rrt	UDPipe	**97.96**	**97.53**	**98.41**	91.31	86.74	82.57	79.02	**81.09**
		UDify	97.73	96.12	95.84	**93.16**	**88.56**	**84.87**	**79.20**	79.92
Russian GSD	ru_gsd	UDPipe	**97.10**	**92.66**	**97.37**	88.15	84.37	82.66	**74.07**	**80.03**
		UDify	96.91	87.45	77.73	**90.71**	**86.03**	**84.51**	67.24	62.08
Russian PUD	ru_pud	UDify	**93.06**	**63.60**	**77.93**	**93.51**	**87.14**	**83.96**	**37.25**	**61.86**
Russian SynTagRus	ru_syntagrus	UDPipe	**99.12**	**97.57**	**98.53**	93.80	92.32	90.85	**87.91**	**89.17**
		UDify	98.97	96.29	94.47	**94.83**	**93.13**	**91.87**	86.91	85.44
Russian Taiga	ru_taiga	UDPipe	93.18	82.87	89.99	75.45	69.11	65.31	48.81	57.21
		UDify	**95.39**	**88.47**	**90.19**	**84.02**	**77.80**	**75.12**	**59.71**	**65.15**
Sanskrit UFAL	sa_ufal	UDify	**37.33**	**17.63**	**37.38**	**40.21**	**18.56**	**15.38**	**0.85**	**4.12**
Serbian SET	sr_set	UDPipe	**98.33**	**94.35**	**97.36**	92.70	89.27	87.08	**79.14**	84.18
		UDify	98.30	92.22	95.86	**95.68**	**91.95**	**90.30**	78.45	**84.93**
Slovak SNK	sk_snk	UDPipe	96.83	**90.82**	**96.40**	89.82	86.90	84.81	74.00	81.37
		UDify	**97.46**	89.30	93.80	**95.92**	**93.87**	**92.86**	**77.33**	**85.12**
Slovenian SSJ	sl_ssj	UDPipe	98.61	**95.92**	**98.25**	92.96	91.16	88.76	**83.85**	**86.89**
		UDify	**98.73**	93.44	96.50	**94.74**	**93.07**	**90.94**	81.55	86.38
Slovenian SST	sl_sst	UDPipe	93.79	86.28	**95.17**	73.51	67.51	63.46	52.67	60.32
		UDify	**95.40**	**89.81**	95.15	**80.37**	**75.03**	**71.19**	**61.32**	**67.24**

Table 4.9: The full test results of UDify on 124 treebanks (part 4 of 5).

Treebank		Model	UPOS	UFeats	Lemmas	UAS	LAS	CLAS	MLAS	BLEX
Spanish AnCora	es_ancora	UDPipe	**98.91**	**98.49**	**99.17**	92.34	90.26	86.39	**83.97**	**85.51**
		UDify	98.53	97.89	98.07	**92.99**	**90.50**	**87.26**	83.43	84.85
Spanish GSD	es_gsd	UDPipe	**96.85**	**97.09**	**98.97**	90.71	**88.03**	**82.85**	**75.98**	**81.47**
		UDify	95.91	95.08	96.52	**90.82**	87.23	82.83	72.47	78.08
Spanish PUD	es_pud	UDify	**88.98**	**54.58**	**100.00**	**90.45**	**83.08**	**77.42**	**18.06**	**77.42**
Swedish LinES	sv_lines	UDPipe	96.78	**89.43**	**97.03**	86.07	81.86	80.32	66.48	77.38
		UDify	**96.85**	87.24	92.70	**88.77**	**85.49**	**85.61**	**66.99**	**77.62**
Swedish PUD	sv_pud	UDify	**96.36**	**80.04**	**88.81**	**89.17**	**86.10**	**85.25**	**57.12**	**72.92**
Swedish Sign Language SSLC	swl_sslc	UDPipe	**68.09**	100.00	**100.00**	**50.35**	**37.94**	**39.51**	**30.96**	**39.51**
		UDify	63.48	**96.10**	**100.00**	40.43	26.95	30.12	23.29	30.12
Swedish Talbanken	sv_talbanken	UDPipe	97.94	**96.86**	**98.01**	89.63	86.61	84.45	79.67	**82.26**
		UDify	**98.11**	95.92	95.50	**91.91**	**89.03**	**87.26**	**80.72**	81.31
Tagalog TRG	tl_trg	UDify	**60.62**	**35.62**	**73.63**	**64.04**	**40.07**	**36.84**	**0.00**	**13.16**
Tamil TTB	ta_ttb	UDPipe	91.05	**87.28**	**93.92**	74.11	66.37	63.71	**55.31**	**59.58**
		UDify	**91.50**	83.21	80.84	**79.34**	**71.29**	**69.10**	53.62	54.84
Telugu MTG	te_mtg	UDPipe	93.07	99.03	**100.00**	91.26	**85.02**	**81.76**	**77.75**	**81.76**
		UDify	**93.48**	**99.31**	**100.00**	**92.23**	83.91	79.92	76.10	79.92
Thai PUD	th_pud	UDify	**56.78**	**62.48**	**100.00**	**49.05**	**26.06**	**18.42**	**3.77**	**18.42**
Turkish IMST	tr_imst	UDPipe	**96.01**	**92.55**	**96.01**	74.19	**67.56**	63.83	**56.96**	**61.37**
		UDify	94.29	84.49	87.71	**74.56**	67.44	**63.87**	49.42	54.10
Turkish PUD	tr_pud	UDify	**77.34**	**24.59**	**84.31**	**67.68**	**46.07**	**39.95**	**2.61**	**32.50**
Ukrainian IU	uk_iu	UDPipe	97.59	**92.66**	**97.23**	88.29	85.25	81.90	**73.81**	79.10
		UDify	**97.71**	88.63	94.00	**92.83**	**90.30**	**88.15**	72.93	**81.04**
Upper Sorbian UFAL	hsb_ufal	UDPipe	62.93	41.10	68.68	45.58	34.54	27.18	**3.37**	16.65
		UDify	**84.87**	**48.84**	**72.68**	**71.55**	**62.82**	**56.04**	16.19	**37.89**
Urdu UDTB	ur_udtb	UDPipe	93.66	81.92	**97.40**	87.50	81.62	75.20	55.02	73.07
		UDify	**94.37**	**82.80**	96.68	**88.43**	**82.84**	**77.00**	**56.70**	**73.97**
Uyghur UDT	ug_udt	UDPipe	**89.87**	**88.30**	**95.31**	**78.46**	**67.09**	**60.85**	**47.84**	**57.08**
		UDify	75.88	70.80	79.70	65.89	48.80	38.95	21.75	31.31
Vietnamese VTB	vi_vtb	UDPipe	89.68	**99.72**	**99.55**	70.38	62.56	60.03	55.56	59.54
		UDify	**91.29**	99.58	99.21	**74.11**	**66.00**	**63.34**	**58.71**	**62.61**
Warlpiri UFAL	wbp_ufal	UDify	**33.44**	**18.15**	**39.17**	**21.66**	**7.96**	**7.49**	**0.00**	**0.88**
Yoruba YTB	yo_ytb	UDify	**50.86**	**78.32**	**85.56**	**37.62**	**19.09**	**16.56**	**6.30**	**12.15**

Table 4.10: The full test results of UDify on 124 treebanks (part 5 of 5).

Conclusion

We have proposed and evaluated UDify, a multilingual multi-task self-attention network fine-tuned on BERT pretrained embeddings, capable of producing annotations for any UD treebank, and exceeding the state-of-the-art in UD dependency parsing in a large subset of languages while being comparable in tagging and lemmatization accuracy. To the best of our knowledge, UDify is the largest multilingual part-of-speech tagger, morphological analyzer, lemmatizer, and dependency parser to date, capable of parsing 75 languages across 124 treebanks. And this is the first self-attention only model providing strong results in part-of-speech tagging and dependency parsing.

Strong regularization and task-specific layer attention are highly beneficial for fine-tuning, while also reducing the number of required models to train down to one by training multilingually. Multilingual learning is most beneficial for low-resource languages, even ones that do not possess a training set and can be further improved by fine-tuning monolingually using BERT weights saved from UDify's multilingual training.

Future Work

There are many more possible avenues to explore in UDify. For one, morphologically, UDify is able to obtain high accuracy in lemmatization and morphology tagging but leaves much to be desired. To make the model as simple as possible, we did not experiment with character-level embeddings. But with character-level features that have been shown to model morphological information well, we predict UDify would surpass UDPipe Future on these metrics. Second, some of the fine-tuned languages result in performance far below the baseline. An improved training strategy that biases the training towards these underperforming languages would likely improve them. For instance, we could randomly select uniform batches from each treebank, and increase the probability of selecting batches of low-resource languages over high-resource ones. Third, we would like to see if BERT fine-tuned on UD can improve the performance of benchmarks on GLUE, MNLI, SQuAD, etc. over using just the unsupervised BERT model. The intuition is that a BERT model biased towards human-annotated syntax can possibly provide better word representations than through unsupervised pretraining alone. Fourth, we can try to extend the model to support XPOS annotations. One simple approach could be to concatenate all XPOS annotations, same as all the other annotations, and then insert a treebank identifier to allow the network to pick the appropriate XPOS annotation scheme for a treebank.

Summary of Results

To summarize, we provide answers to the research questions listed in the introduction.

1. *What specific architectural choices are necessary for building a model capable of scaling to annotate Universal Dependencies across all supported languages?*

   Self-attention networks demonstrate a remarkable capability to differentiate between many languages without harming evaluation performance. To provide

accurate annotations, a pretraining strategy like BERT is necessary to bias the attention weights across the majority of the considered languages. To prevent further overfitting to any individual language, regularization techniques are crucial, i.e., increased dropout, discriminative fine-tuning, inverse square root learning rate decay, input masking, task-specific layer attention, and layer dropout. Combined, these strategies produce a model capable of surpassing UDPipe Future, a strong state-of-the-art baseline that that must train a model per treebank.

2. *How does multilinguality affect the performance of parsing Universal Dependencies, and which languages benefit the most from cross-lingual annotations?*

   With respect to UDify, multilingual pretraining and fine-tuning often produce superior annotations when compared to language-specific fine-tuning, especially if the languages are low-resource and are similar to other provided languages. Slavic languages tend to have the largest benefit, as there are many supported Slavic languages provided by UD, and they are all linguistically similar to one another.

3. *In what ways does unsupervised pretraining help for training self-attention networks for dependency parsing?*

   Pretraining self-attention networks introduce a strong syntactic bias that is capable of generalizing across languages. This is evidenced by experiments that suggest that pretraining improves dependency parsing performance not only for pretrained languages but also on languages that were never pretrained to begin with. Furthermore, this shows that even simple multilingual pretraining strategies that do not explicitly define cross-lingual training tasks can still result in cross-lingual word representations that result in more accurate multilingual dependency parsing. A model that is pretrained on a large collection of languages is very likely to provide useful syntactic information when encountering a completely new language, whether it be in pretraining, fine-tuning, or inference.

4. *What evidence is there to support why pretrained self-attention networks excel in dependency parsing?*

   Evidence provided by a syntactic probe suggests self-attention networks encode structural dependencies between words as a linearly-separable global property of the vector space. A visualization of the minimum spanning tree of a linear function of the L2 norm/distance between the word vectors shows a striking similarity to human-annotated dependency trees, and these get more similar with fine-tuning. Further analysis of the attention heads reveals attention patterns that appear to identify syntactic properties of the input sentence like constituent/phrasal boundaries, and these properties are more prominent after fine-tuning. This evidence all points to the fact that self-attention is remarkably capable of finding structural patterns in text, i.e., syntax, and closely mimics the structural properties of dependency trees when processed through multiple layers.

Bibliography

Salim Abu-Rabia and Ekaterina Sanitsky. Advantages of bilinguals over monolinguals in learning a third language. *Bilingual Research Journal*, 33(2):173–199, 2010.

Wasi Uddin Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. On difficulties of cross-lingual transfer with order differences: A case study on dependency parsing. *arXiv preprint arXiv:1811.00570*, 2018.

Jay Alammar. The illustrated transformer. 2018. URL `http://jalammar.github.io/illustrated-transformer/`.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A Smith. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444, 2016.

Mikel Artetxe and Holger Schwenk. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *arXiv preprint arXiv:1812.10464*, 2018.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. Cloze-driven pretraining of self-attention networks. *arXiv preprint arXiv:1903.07785*, 2019.

Miguel Ballesteros, Chris Dyer, and Noah A Smith. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, 2015.

Toms Bergmanis and Sharon Goldwater. Context sensitive neural lemmatization with lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 1391–1400, 2018.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoNLL 2016*, page 10, 2016.

Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Grzegorz Chrupała. Simple data-driven context-sensitive lemmatization. *Procesamiento del Lenguaje Natural*, 37, 2006.

Yoeng-Jin Chu. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14: 1396–1400, 1965.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Kevin Clark, Minh-Thang Luong, Christopher D Manning, and Quoc V Le. Semi-supervised sequence modeling with cross-view training. *arXiv preprint arXiv:1809.08370*, 2018.

Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 400–407. ACM, 2005.

Miryam de Lhoneux, Johannes Bjerva, Isabelle Augenstein, and Anders Søgaard. Parameter sharing between dependency parsers for related languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4992–4997, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1723–1732, 2015.

Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

Timothy Dozat, Peng Qi, and Christopher D Manning. Stanford's graph-based neural dependency parser at the conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, 2017.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. 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)*, volume 2, pages 845–850, 2015.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

Jack Edmonds. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240, 1967.

Stephen Fagan and Ramazan Gençay. An introduction to textual econometrics. *Handbook of empirical economics and finance*, pages 133–154, 2011.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, 2016.

John R. Firth. A synopsis of linguistic theory. *1930-55*, 1957.

Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2): 23–38, 1994.

Yoav Goldberg. Assessing bert's syntactic abilities. *arXiv preprint arXiv:1901.05287*, 2019.

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. Effective strategies in zero-shot neural machine translation. *arXiv preprint arXiv:1711.07893*, 2017.

Jirka Hana. Intro to linguistics: Syntax 1. *UFAL*, 2011.

Georg Heigold, Guenter Neumann, and Josef van Genabith. An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 505–513, 2017.

Daniel Hershcovich. Non-projective example. 2017. URL `https://linguistics.stackexchange.com/questions/3640/non-projective-example`.

John Hewitt and Christopher D Manning. 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*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 328–339, 2018.

Go Inoue, Hiroyuki Shindo, and Yuji Matsumoto. Joint prediction of morphosyntactic categories for fine-grained arabic part-of-speech tagging exploiting tag dictionary information. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 421–431, 2017.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1681–1691, 2015.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al.

Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. 2019. URL `http://cs231n.github.io/neural-networks-1/`.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. *arXiv preprint arXiv:1805.01052*, 2018a.

Nikita Kitaev and Dan Klein. Multilingual constituency parsing with self-attention and pre-training. *arXiv preprint arXiv:1812.11760*, 2018b.

Daniel Kondratyuk, Tomáš Gavenčiak, Milan Straka, and Jan Hajič. Lemmatag: Jointly tagging and lemmatizing for morphologically rich languages with brnns. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4921–4928, 2018.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436, 2015.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dekang Lin and Patrick Pantel. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(4):343–360, 2001.

Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luis Marujo, and Tiago Luis. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, 2015.

Yichao Lu, Phillip Keung, Faisal Ladhak, Vikas Bhardwaj, Shaonan Zhang, and Jason Sun. A neural interlingua for multilingual machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 84–92, 2018.

Ryan McDonald, Slav Petrov, and Keith Hall. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of the conference on empirical methods in natural language processing*, pages 62–72. Association for Computational Linguistics, 2011.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Phoebe Mulcaire, Jungo Kasai, and Noah Smith. Polyglot contextual representations improve crosslingual transfer. *arXiv preprint arXiv:1902.09697*, 2019.

Thomas Müller, Ryan Cotterell, Alexander Fraser, and Hinrich Schütze. Joint lemmatization and morphological tagging with lemming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2268–2274, 2015.

Tahira Naseem, Regina Barzilay, and Amir Globerson. Selective sharing for multilingual dependency parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 629–637. Association for Computational Linguistics, 2012.

Joakim et al. Nivre. Universal dependencies 2.3, 2018. URL `http://hdl.handle.net/11234/1-2895`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Harvard NLP. The annotated transformer. 2018. URL `http://nlp.seas.harvard.edu/2018/04/03/attention.html`.

Christopher Olah. Understanding lstm networks. 2015. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

Martin Popel and Ondřej Bojar. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70, 2018.

Cicero D Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

Rico Sennrich and Barry Haddow. Linguistic input features improve neural machine translation. *arXiv preprint arXiv:1606.02892*, 2016.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Damien Sileo. Understanding bert transformer: Attention isn't all you need. *Towards Data Science*, 2019.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123, Brussels, Belgium, October 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K18-2011`.

Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. 2018.

Tobias Sterbak. Guide to sequence tagging with neural networks in python. 2017.

Milan Straka. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium, October 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K18-2020`.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, 2018.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kübler, Marie Candito, Jennifer Foster, Yannick Versley, Ines Rehbein, and Lamia Tounsi. Statistical parsing of morphologically rich languages (spmrl): what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12. Association for Computational Linguistics, 2010.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Jesse Vig. Visualizing attention in transformer-based language models. *arXiv preprint arXiv:1904.02679*, 2019.

David Vilares, Carlos Gómez-Rodríguez, and Miguel A Alonso. One model, two languages: training bilingual parsers with harmonized treebanks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 425–431, 2016.

Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Wen-tau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question answering using enhanced lexical semantic models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1744–1753, 2013.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

Samy Zafranyj. Nlp with gensim (word2vec). 2019. URL `https://samyzaf.com/ML/nlp/nlp.html`.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium, October 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/K18-2001`.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1213–1222, 2015.

George Kingsley Zipf. Human behavior and the principle of least effort. 1949.

List of Figures

List of Tables