Master's Thesis

# Character-level Neural Architectures for Jointly Predicting Word Alignments and Word-internal Structure in Morphologically Complex Languages

*Author:*
Sander Bijl de Vroe

*External Supervisor:*
Dr. Wilker Ferreira Aziz

*Primary Supervisor:*
Dr. Rick Nouwen

*Secondary Supervisor:*
Prof. Dr. Jan Odijk

Utrecht University

September 12, 2017

**Abstract**

Through a word alignment task between English and Turkish, this project investigates ways to more effectively approach morphologically complex languages in the field of Natural Language Processing (NLP). Our models create an inductive bias to focus on word-internal structure, by taking character-level input and jointly predicting alignment, lemmas and morphological tags. Current versions of the model are able to exploit the lemma distribution so that the predicted alignment distribution improves in quality, while possible improvements to the morphological tag side of the architecture are identified. Furthermore, different methods of encoding character-level input are explored, suggesting that modern neural architectures might benefit from using multiple types of encoders in conjunction. Finally, the benefit of moving away from word-level input data towards the character level is further supported.

# 1 Introduction

The current project explores ways to deal with word-internal structure in a word alignment task between Turkish and English. Finding ways to acknowledge word-internal structure is an important unanswered problem in NLP. Previously, tasks were tackled almost exclusively at the word level, leading to a variety of issues. These issues are related to data sparsity, to building fixed, limited vocabularies, and to the models' ignorance of identical internal constituents of words. The prevalence of such problems is exacerbated in morphologically complex languages, where the ability to produce many different forms of words leads to much larger vocabularies of surface forms.

Recently, various models have been proposed to deal with these issues. A promising avenue of research is treating input at the character level (Kim et al., 2016), while feeding input as units resembling morphemes has also shown encouraging results (Vylomova et al., 2016).

Word alignment is a task that would greatly benefit from word-internal knowledge, and Turkish is a typical example of a morphologically complex language. We present a neural architecture to predict word alignments, and firstly explore various character-level encoders, based on a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN). Secondly, we investigate whether it is possible to create an inductive bias in these encodings, by letting them predict not only word alignments, but also lemmas and morphological features. We also present a word-level baseline that has no access to word-internal structure, which serves to elucidate the benefit of moving to character-level input.

Our results indicate that it is possible to bias the model with improved results on word alignment. However, this benefit only occurs for the model with lemma information, and under the current design morphological features present a distraction. Possible issues with the architecture are considered, and solutions concerning the convolutional encoder and the morphological data are proposed. Furthermore, future directions that are able to deal with the noisy input data (such as the Variational Autoencoder (VAE)) are suggested, along with other avenues for research.

Section 2 will explore a number of background topics that will be essential to understanding the thesis. Morphological complexity and its consequences in Natural Language Processing will be discussed, and the problem of word aligment will be explained. Before setting forth our solution, previous translation and alignment models will be introduced, together with a number of neural network architectures involved in the implementation. Section 3, then, presents the current project's main model and its variants, while section 4 explains the experimental setup and the data involved. Sections 5 and 6 analyze and discuss the results. Code and data for this project are available on `https://github.com/Bednas/thesis_code.git`.

# 2 Background

## 2.1 Morphological complexity

Languages can be classified into three different types based on the transparency of their word-internal semantic boundaries. In an isolating language, there are no word-internal boundaries, and each word is thus typically built up of a single morpheme (Aikhenvald, 2007). Agglutinative languages, on the other hand, build words with clear word-internal boundaries; a word may consist of multiple morphemes that represent a single morpho-syntactic property. Finally, fusional languages have vague word-internal boundaries. Like agglutinative languages, their words may contain mul-

tiple morphemes, but more than one morpho-syntactic feature may be grouped into each unit[1].

For instance, consider the way Latin (a typical fusional language) and Turkish (a typical agglutinative language) construct the phrase 'of houses', presented in table 1.

| | Latin | | Turkish | | |
|---|---|---|---|---|---|
| Word | domuum | | evlerin | | |
| Morphemes | dom | uum | ev | ler | in |
| Translation | house | PL, GEN | house | PL | GEN |

Table 1: Fusional vs. agglutinative word formation. 'PL' is short for plural, a marker for number. 'GEN' is short for genitive, a case marker indicating possession.

Agglutinative and fusional languages thus exhibit considerable morphological complexity. Words in these languages may contain many separate morpho-syntactic features at once, and agglutinative languages in particular can contain many morphemes. Furthermore, since it is possible to combine multiple morphemes to form a word, there are many more different word forms than in an isolating language.

Languages can also be categorized as belonging to the analytic, synthetic or polysynthetic types (Aikhenvald, 2007). This distinction revolves around the number of morphemes per word; whereas an analytic language has very few morphemes per word (approaching a one-to-one correspondence), a synthetic language uses words that typically consist of multiple morphemes. Polysynthetic languages have extreme internal complexity to their words, such that a single word often corresponds to entire clauses. How a language falls on these scales has important consequences for NLP systems. Before considering the problems involved with morphological complexity, however, we will first consider English and Turkish in terms of their morphology.

## 2.2 Turkish and English Morphology

Turkish is a language spoken by about 88 million speakers, with most of its L1 speakers in Anatolia and Southeastern Europe. As mentioned previously, it is a typical example of a language with agglutinative morphosyntax, and it makes use of extensive suffixing (Kuribayashi, 2013). This process can form enormous words, equivalent to entire English sentences. This is well illustrated by a word borrowed from Lewis (1970): 'avrupalılaştırılamıyabilenlerdenmişsiniz', which translates to 'You seem to be one of those who may be incapable of being Europeanized'.

Turkish has clear but complex restrictions on morpheme ordering. Its morphotactics can be modeled by a finite state transducer, and the word produced by a set of morphemes is deterministic (Oflazer, 1994). However, there are often multiple possible morphological analyses for a given surface form. Another challenge is Turkish vowel harmony. Vowels in a word change according to the vowels that occur before them, so that most morphemes have more than one possible instantiation depending on their context (Oflazer, 1994).

English is morphologically speaking quite different from Turkish. Its morphosyntax is considerably less complex, putting it closer to an isolating language. It is often categorized as an analytic language; its ratio of morphemes to words is low (Konig and Van der Auwera, 2013). English has lost most of its inflectional morphology over time, with invariant pronouns and adjectives, and

[1]It is worth noting here that both typologies presented in this section are not viewed as having entirely distinct categories, but rather as being on a scale (Aikhenvald, 2007).

almost no case inflection on the noun phrase. Its verb inflections have also been greatly reduced, as it has lost many distinctions in mood, person and number. English is considerably richer in its derivational morphology than in its inflectional morphology, using various prefixes and suffixes to build new words (Carstairs-McCarthy, 2002). In comparison to an agglutinative language like Turkish, there is considerably less morphological ambiguity in an analytic language, although there are some cases of ambiguity from this derivational morphology in English (i.e. (Vikner and Vikner, 2008)).

## 2.3 NLP Solutions to Complex Morphology

The morphological properties as described in the previous sections have ramifications for how an NLP system deals with a language. For isolating or analytic languages such as English, word-level treatment is reasonable, since the most basic unit of meaning often coincides with the word. However, many other common languages with more complex morphologies run into problems. A first concern is that, through word-level treatment, the internal structure of words is obscured, causing linguistic information to be lost. For an agglutinative language, many of these more basic units of meaning are no longer visible to the system. For example, it is no longer immediately apparent to these approaches that two words that share an internal constituent share a morpho-syntactic property.

There are also practical reasons to avoid word-level treatment. With agglutination, vocabulary sizes can become enormous or even unbounded, since it allows for the construction of far more word forms than is typical of an isolating language. In most word-level applications there is a limit to the vocabulary size due to technical limitations. The most frequent words are kept in the vocabulary, and rare words are all labeled as unknown ('UNK'). When a language is agglutinative, this tail of the word frequency distribution becomes more significant, and solving an NLP task becomes much harder because the identity of many of the input words is lost.

A related problem is that of data sparsity. With many more surface forms, models will have to estimate a larger number of parameters. Reliable estimation of large parameter sets will require many more observations, and it can be difficult for a statistically trained word-based model to see enough data. This becomes particularly hard when modeling multinomial distributions, depending on the model. For n-gram language models, for instance, the complexity is exponential, with $\mathcal{O}(|\mathcal{V}|^n)$, where $\mathcal{V}$ is the vocabulary. With data sparsity it becomes far more difficult to estimate a probability for rare words, and it becomes more likely for the model to encounter words in the input that its training corpus did not even contain. For such out-of-vocabulary words it is even harder to estimate a probability.

To circumvent issues like the ones mentioned above, a number of approaches have been proposed, most of which attempt to decompose the word into smaller units. For instance, one solution in Statistical Machine Translation (SMT) has been to separately train two models; one of these is a normal SMT model and the other predicts the fully inflected form of each target word (Toutanova et al., 2008). Inflection prediction is performed using different types of information, including the target stem, morphological analyses of both sides and syntactic information. Thus, the model gains access to more linguistic information and fine-grained units, improving the model's performance.

Another solution to tackle agglutinative languages in SMT is to train the system on morphemic input instead of whole words (Bisazza and Federico, 2009). Various morphological segmentation schemes are applied to the data, which reduce the size of the vocabulary and allow the model to attain higher translation scores. This approach is a less supervised model than the previous,

since it does not include linguistic information in the source aside from segmentation. A few similar attempts to move to smaller units of meaning have been made. For instance, in an English to Turkish word alignment task Çakmak et al. (2012) segment the Turkish side into stems and suffixes, leading to a large jump in performance. Such a step has also been made in Neural Machine Translation (NMT); by switching to translating subword units when encountering rare or out-of-vocabulary words, the system performs better (Sennrich et al., 2015). This approach is even less linguistically supervised than the previous, as the optimal segmentation scheme was based on a data science compression algorithm called byte pair encoding instead of on linguistic knowledge.

Finally, there have been attempts to move to even lower levels, allowing models to take characters as their input. For instance, an entirely unsupervised language modeling network that uses only characters was proposed, which outperformed both word-based and morpheme-based baselines (Kim et al., 2016). Using this type of representation evades the out-of-vocabulary problem; the character vocabulary is not open, so the model will never encounter an unseen token. The design of such networks will be described later, as we first explain this project's task.

## 2.4 Word Alignment

The problem we intend to tackle in this project is that of word alignment. In word alignment, the goal is to find translational equivalence between words in two sentences that have the same meaning (Smith, 2011) An example taken from the validation corpus is shown in figure 1. Word alignments have been useful to many different types of problems in NLP. Most notably they have been a vital part of SMT, from early models (Brown et al., 1988) to later phrase-based hierarchical models (Chiang, 2005). Word alignment has also proven useful to semantic role labeling; making it possible to project semantic roles in one language onto another language using alignment information (Padó and Lapata, 2009). In general, they are useful in leveraging information from one language to another, and as such they have also been useful in creating bilingual word embeddings, learning about a language's structure through a parallel corpus and even for modeling lexical changes over time (Smith, 2011).

We therefore focus on word alignment because of its worth to fields such as machine translation and NLP in general. However, word alignment was also chosen for more practical reasons, since a word alignment model takes less time to implement than a full machine translation pipeline, making it more suited for a project of this length.

Alignment is typically more formally defined as follows. Take a sentence $x_1^m \in \mathcal{V}_\mathcal{X}^*$ and a sentence $y_1^n \in \mathcal{V}_\mathcal{Y}^*$ such that $x_1^m$ and $y_1^n$ are translations of each other, $m$ and $n$ are the lengths of the sentences measured in occurrences of word forms (tokens), and $\mathcal{V}_\mathcal{X}$ and $\mathcal{V}_\mathcal{Y}$ are the vocabularies of languages $\mathcal{X}$ and $\mathcal{Y}$, in which the respective sentences are writen. The task is to find the correspondences between words that are each other's translation in this context. A set of all correspondences between words in $x_1^m$ and $y_1^n$ is a set $a \in 2^{\{1,2,\ldots,m\}\times\{1,2,\ldots,n\}}$ (Smith, 2011); By this general definition, a word from either $x_1^m$ or $y_1^n$ may align to multiple other words. If a word does not align with any word in the other sentence it is simply excluded from the set of relations $a$. The models we later propose will change this definition slightly, as restrictions on the number of alignments will be placed on one side.

In solving alignment through machine learning, the task should become easier if the model has access to morphological information. Often it may be a part within a word that aligns to a word in the other sentence, so a model that is better informed of morphology should better be able to relate a complex word to all of its counterparts in the other language. For instance, in the example

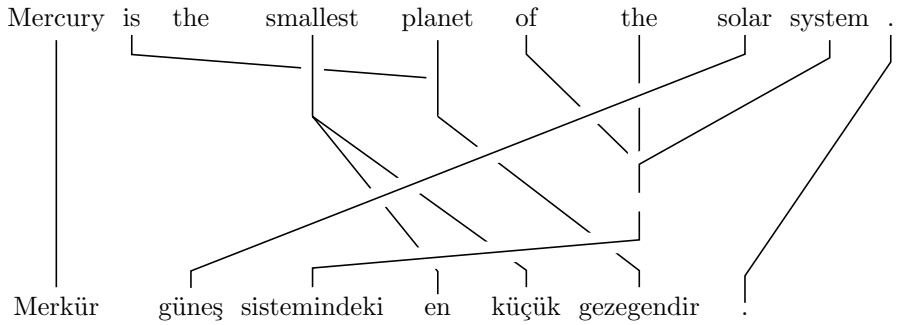



Figure 1: Word alignments between an English and a Turkish sentence. In terms of a set of correspondences this can be expressed as the set $\{\langle 1,1\rangle, \langle 6,2\rangle, \langle 4,4\rangle, \langle 5,4\rangle, \langle 6,5\rangle, \langle 3,6\rangle, \langle 3,7\rangle, \langle 2,8\rangle, \langle 3,9\rangle, \langle 7,10\rangle\}$, where the first integer is a position in the Turkish sentence, and the second integer is a position in the English sentence.

in figure 1, 'sistemindeki' aligns to 'of', 'the' and 'system'. In this case, a deeper understanding of the morphological content of 'sistemindeki' could help generate the correct alignments.

The quality of a predicted set of alignments will in this project be calculated by the Alignment Error Rate (AER) (Och and Ney, 2003), which is often used to measure performance of word alignment. The AER is derived from the F-measure (combined from redefined recall and precision measures), and is defined as follows:

$$AER(S, P; A) = 1 - \frac{|A \cap S| + |A \cap P|}{|A| + |S|}, \tag{1}$$

Where $S$ is the set of sure alignments, $P$ is the set of possible alignments, and $A$ is the predicted set of alignments that is being compared[2]. Note that the alignment data in this project does not contain possible alignments, and thus all alignments will be considered sure. In AER's definition an alignment that is considered sure is also possible ((Och and Ney, 2003)), so the formula can in our case be adapted as follows:

$$AER(S; A) = 1 - \frac{2|A \cap S|}{|A| + |S|}, \tag{2}$$

## 2.5 IBM Models

Alignment prediction in this project is heavily inspired by early translation and alignment work (Brown et al., 1988, 1993). The IBM models are a series of models of increasing complexity that can be used for translating sentences, taking into account word alignments.

The probabilistic graphical model that they propose can be depicted as in figure 2. In discussing the model we take the source language to be Turkish and the target language English, as will be the case in this project's task.

[2]Sure and Possible alignments are used for human annotation of gold-labeled data to indicate ambiguous cases

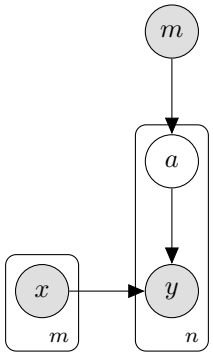



Figure 2: IBM Models 1 and 2

$X$ and $Y$ are both observed categorical random variables taking words from respectively the Turkish and the English vocabularies $\mathcal{V}_x$ and $\mathcal{V}_y$. Let $A$ be a latent categorical random variable taking alignments. In our model, $A$ is dependent on $M$, an observed categorical random variable representing the length of the Turkish sentence.

A Turkish sentence is a random sequence of $m$ Turkish words, which we denote $X_1^m = \langle X_1, ..., X_m \rangle$. The English sentence is a random sequence of $n$ English words, denoted as $Y_1^n = \langle Y_1, ..., Y_n \rangle$. Finally, corresponding to each assigned word $y_j$ in this sentence we have an alignment $a_j$, which is an integer between 0 and $m$ that represents the index in the Turkish sentence for the Turkish word that generated the English word[3]. The basic goal behind this translation model is to predict a sequence of English words $Y_1^n$. Each of these is the translation of a word in the Turkish sentence, and we know which word to translate by considering word alignments.

In order to predict translations, then, we need to model the probability distribution $P(Y_1^n = y_1^n, A_1^n = a_1^n | X_1^m = x_1^m, M = m)$. This distribution can be factorized according to the independence assumptions implied by the probabilistic graphical model[4]:

$$P(Y_1^n = y_1^n, A_1^n = a_1^n | X_1^m = x_1^m, M = m) = \prod_{j=1}^{n} P(y_j, a_j | x_1^m, m) \tag{3}$$

$$= \prod_{j=1}^{n} P(a_j | m) P(y_j | x_{a_j}) \qquad \text{IBM assumptions} \tag{4}$$

IBM models 1 and 2 make a few assumptions here. First, we assume that an alignment link $(a_j)$ depends only on $m$ and that $y_j$ depends only on the word $x_{a_j}$, the word it aligns to in $x_1^m$. $x_{a_j}$ is selected through the alignment link $a_j$, so that a single word $y_j$ no longer depends on the entire sentence $x_1^m$. In a way, the alignment $a_j$ shows us where in the sequence $x_1^m$ to look in order to correctly translate word $y_j$. Note that the model therefore assumes that each target word

[3]Index 0 represents the null alignment, which occurs when a word isn't aligned to any word in the translated sentence.

[4]For conciseness, we only write out the random variable's assignment in the first equation. Thus $Y_1^m = y_1^m$ is after its first mentioning referred to as simply $y_1^m$.

is generated by a single source word. This assumption is often violated in reality; consider the English word 'not', which may be generated by two French words 'ne' and 'pas'. These particular models also assume that the length of the target sentence is known in advance.

The lexical distribution $P(Y|X = x)$ is parameterized by a categorical distribution, that is $Y|x \backsim \text{Cat}(\theta_x)$, where $\theta_x \in \mathbb{R}^{\mathcal{V}_y}$, $0 \leq \theta_{x,y} \leq 1$ and $\sum_y \theta_{x,y} = 1$. In implementation this can be thought of as a large lookup table of parameters, with a separate value for a word in the vocabulary of $y$ given each word in the vocabulary of $x$.

The alignment distribution can be parameterized in different ways. For IBM model 1, $P(A|M = m)$ is parameterized by $\mathcal{U}(\frac{1}{m+1})$, where $\mathcal{U}$ is the uniform distribution. A priori this implies that, independent of the words involved, any position in one sentence may align to any other position in the other sentence with equal probability. Thus word 1 in a sentence is just as likely to align to the first word as to the last word in the translation. This is a simplification - between many languages, especially pairs with similar word orders, a word is more likely to align to words in similar positions in another sentence.

Other IBM models build on this first description, and loosen some of its restrictive assumptions in various ways. For instance, IBM model 2 allows more flexibility for the alignment distribution, utilizing a categorical parameterization. This leads to a heavily overparameterized model, and various solutions have been proposed to estimate the parameters (i.e. 1996). The original paper (Brown et al., 1988) describes a total of 5 models, each of which is more complex than the previous. The current project, however, predicts word alignments in a similar way to IBM Model 1. In essence, model 1 focuses most on the lexical distribution due to its simplifying assumptions on the alignment distribution. The investigation in this project induces biases through the lexical distribution, so in choosing model 1 we ensure that improvements are more visible.

As will be described later, the parameters of the categorical lexical distribution will be predicted by a neural network, however the assumptions involved in the alignment and lexical distributions remain the same. In order to explain the implementation of this network, a number of neural architectures will now be described.

## 2.6 Modern NLP Architectures

### 2.6.1 Embeddings

Word embeddings have become essential as representations for words in recent years (Mikolov et al., 2013). An embedding is a continuous vector representation. Words that have embeddings that are close to each other in this vector space tend to behave similarly; it can be empirically shown that embeddings produced by certain training algorithms capture lexical, syntactical or semantic information (Goldberg, 2016). Various types of neural networks are used to train embeddings. One popular model is the continuous skip-gram model, which learns vectors for words that best predict the occurrence of other words in the same sentence (Mikolov et al., 2013).

The concepts behind word embeddings can also be used for characters to create a character-level representation of a word. Suppose we are given a word $w \in \mathcal{V}$, which is a sequence of characters $[c_1, ..., c_{k_c}]$, where $c \in \mathcal{C}$, the vocabulary of characters, and $k_c$ is the length of word $w$ in characters. Each character in vocabulary $\mathcal{C}$ can be embedded into a dense vector representation $\in \mathbb{R}^{d_c}$ as described above (Mikolov et al., 2013). Given the matrix of these character embeddings $\mathbf{Q} \in \mathbb{R}^{d_c \times |\mathcal{C}|}$, we can create a character-level representation $\mathbf{C}^w \in \mathbb{R}^{k_c \times d_c}$ for word $w$. In matrix $\mathbf{C}^w$, the $i$-th column is the character embedding for character $c_i$ (taken from matrix $\mathbf{Q}$).

There are multiple ways to subsequently transform matrix $\mathbf{C}^w$into a vector resembling a word embedding. Such techniques allow us to arrive all the way from the character-level at a word representation. Two recently proposed methods of doing so encode the matrix using an RNN and a CNN, which will be proposed in sections 2.6.3 and 2.6.4. We first present a more basic neural architecture.

### 2.6.2 Multilayer Perceptron

Multilayer Perceptron (MLP)s belong in the category of feedforward artificial neural networks, and consist of a layer of input nodes, a number of layers of hidden nodes, and a layer of output nodes. For each node after the first layer, the inputs are multiplied by a learned weight (with an optional learned bias term added) and the result is summed before being passed through a nonlinear activation function. An MLP is called densely connected if each node after the first layer receives input from all of the nodes in the layer before. A network with enough of these connected nodes, even with only one hidden layer of nodes and with specific activation functions, has been proven to be capable of approximating any continuous function (Cybenko, 1989).

The activation functions can be considered to resemble the firing rate of a biological neuron. Under this abstraction, the node can be understood to 'fire' if its combined input is sufficiently high. Examples include the logistic sigmoid function and the hyperbolic tangent function, both of which are used in this project. Another such function which is useful to classification tasks is the Softmax function:

$$\text{softmax}(\mathbf{v})_j = \frac{e^{v_j}}{\sum_{k=1}^{K} e^{v_k}} \text{ for j} = 1, \ldots, \text{K}. \tag{5}$$

Here $\mathbf{v}$ is a vector. For each value in the vector, the softmax function outputs a new value in the range $[0, 1]$, such that the sum of all values in the new vector is 1. In this sense, the coordinates of the vector resemble the parameters of a categorical probability distribution. Note that this function depends on the output of multiple nodes; each element in the vector corresponds to a single node in the same layer of the network.

An MLP can learn to perform a task through training on large datasets. Its parameters are first randomly initialized and then updated such that the difference between a training example's value and the network's output is minimized. Parameter optimization is typically performed using variants of stochastic gradient descent and the backpropogation algorithm. MLPs, and variations on their architecture, have been used to tackle many different problems in NLP with strong results (Goldberg, 2016).

### 2.6.3 Recurrent Neural Networks

RNNs are a neural network architecture, in which the input is treated as a sequence, and each hidden state has access to the previously calculated hidden state. This acknowledges that one input from the sequence may depend on inputs that have occurred at previous timesteps. In contrast to MLPs, RNNs are thus allowed to be recursive. More formally, whereas first each hidden state was calculated by

$$h_t = f(v_t), \tag{6}$$

in an RNN they are calculated by

$$h_t = f(h_{t-1}, v_t). \tag{7}$$

Various proposals have been made concerning the definition of the nonlinear function f in the previous equation. Standard activation functions struggled with the problem that the learning signal became increasingly small as sequence length increased, due to many multiplications of small-sized gradients. Thus, weights associated with the beginning of the sequence received negligible updates, and the network would be unable to capture long-range dependencies (Pascanu et al., 2013).

This so called vanishing gradient problem was first addressed by the Long Short-Term Memory (LSTM) unit (Hochreiter and Schmidhuber, 1997). The current project uses Gated Recurrent Unit (GRU), an architecture inspired by the LSTM that achieves a similar solution but uses fewer parameters in the process (Cho et al., 2014). Gated recurrent units incorporate a reset gate and an update gate, which allow it to adaptively choose what to remember and forget. The gates are defined as follows:

$$\text{Reset: } r_j = \sigma([W_r v]_j + [U_r h_{\langle t-1 \rangle}]_j), \tag{8}$$

$$\text{Update: } z_j = \sigma([W_z v]_j + [U_z h_{\langle t-1 \rangle}]_j). \tag{9}$$

Where $\sigma$ is the softmax function, $[\cdot]_j$ is the $j^{th}$ element of a matrix, $W$ and $U$ are weight matrices that are updated in the training process, $v$ is the input to the state and $h_{\langle t-1 \rangle}$ is the value of the previous hidden state. The gates are used in conjunction to calculate the hidden state as follows:

$$h_j^{\langle t \rangle} = z_j h_j^{\langle t-1 \rangle} + (1 - z_j)\tilde{h}_j^{\langle t \rangle}, \text{ where} \tag{10}$$

$$\tilde{h}_j^{\langle t \rangle} = \phi([Wv]_j + [U(r \odot h_{\langle t-1 \rangle})]_j). \tag{11}$$

Here $\odot$ is the Hadamard product (also known as the elementwise multiplication function), and $\phi$ is a nonlinear activation function. The way these gates are combined in the hidden state calculation allows the reset gate, when it is close to zero, to forget information that it learns will later be irrelevant. The update gate is able to control how much information flows to next state, which in turn lets the RNN remember long-term information. This representation allows the RNN to work effectively with sequence-based tasks with sequences of variable length, in which long-range dependencies are important. This type of task coincides with many natural language processing problems (Goldberg, 2016).

Two possible variants warrant mention here. Firstly, the RNN can either return the output of every hidden state, or return only the output of the final hidden state, depending on what the problem requires. Secondly, it is frequently useful to utilize two RNNs in opposing directions. The input sequence is then presented to the network both forwards and backwards, and the two resulting outputs are typically concatenated. This allows each state to effectively have access to both the subsequent and the preceding hidden states, which can be informative for NLP problems. The resulting architecture is often referred to as a bidirectional RNN, or a bi-RNN.

Recently bidirectional LSTMs have been proposed as a way of creating a word vector from a character embedding matrix (Ling et al., 2015). Ling et al. use a bidirectional LSTM that takes a sequence of character embeddings (like $\mathbf{C}^w$). The bi-LSTM consists of two LSTMs that each take their input in a different direction; each of the networks outputs its final hidden state, and these are combined to arrive at a word representation $\mathbf{y}^w \in \mathbb{R}^h$, where $h$ can be chosen as part of the architecture's hyperparameters. As such, the architecture arrives at separate vectors for each word. The current project adopts a similar method for arriving at a word representation, but uses a bi-GRU instead of the LSTMs.

### 2.6.4 Convolutional Neural Networks

CNNs were first proposed by LeCun et al. (1990). CNNs apply convolutions between a filter (also called a kernel) and a local selection of the input. Convolutions are applied over the entire input, so that a feature map is created, which describes the presence of a local feature in each part of the input. CNNs will have many such filters, so that each filter can be trained to detect a different feature that is valuable for the task.

CNNs' first successes came in the field of image recognition, for which they were originally designed, but they have since been adapted to perform well on other problems. Recently CNNs have been introduced as a method to create semantic representations from characters, diverging from the classical approaches that use the word as their basic linguistic unit. For instance, convolutions over characters have succesfully been applied to text classification (Zhang et al., 2015), machine translation (Lee et al., 2016; Gehring et al., 2016) and general language modeling (Kim et al., 2016).

The encoder used in this project is inspired by Kim et al's network. It obtains feature maps $\mathbf{f}^w \in \mathbb{R}^{k_c - s + 1}$ by applying narrow convolutions between learned filters $\mathbf{H} \in \mathbb{R}^{d_c \times s}$ and a word $k$'s character-level representation $\mathbf{C}^w \in \mathbb{R}^{d_c \times k_c}$, where $k_c$ is the length of the word measured in characters, $s$ is the size (or width) of the filter, $d_c$ is the dimensionality of the character embedding, and $\mathbf{C}^w$is derived as explained in section 2.6.1. The $i$-th element is then calculated as follows:

$$\mathbf{f}^w[i] = \tanh(\langle \mathbf{C}^w[*, i : i + s - 1], \mathbf{H} \rangle + b) \tag{12}$$

Here, the expression in square brackets selects columns $i$ to $i + s - 1$ (the relevant character embeddings in the word) from $\mathbf{C}^w$. tanh is the hyperbolic tangent function and $\langle A, B \rangle$ is the Frobenius inner product of $A$ and $B$.

We then incorporate a max-over-time pooling layer, such that the output corresponding to a filter $\mathbf{H}$ is

$$y^w = \max_i \mathbf{f}^w[i]. \tag{13}$$

We can apply any number of filters with varying widths to a word, each producing a single feature $y^w \in \mathbb{R}$. The output from the convolutional encoder for a single word is a vector $\mathbf{y}^w = [y_1^w, ..., y_h^w], \in \mathbb{R}^h$, where $h$ is the number of filters we choose to apply. This vector is an encoding of word $w$. Note that this method treats each word separately; some other approaches choose to treat the sentence as one long stream of characters, with the space as just another character in the vocabulary. This method becomes our second way of arriving at a word representation from the character level. Technical details of how these word representations are further used to predict alignments will be explained after setting out the project goals and model.

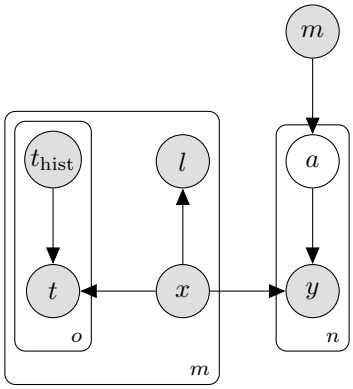



Figure 3: PGM

# 3 The Current Project

## 3.1 Goals

The current project attempts to answer two main questions:

- Given that in our input we are dealing with a language like Turkish that has an essentially open vocabulary, how can we best encode our input?
- Is it useful for the alignment task to bias the resulting representations, by simultaneously predicting morphological tags and the lemma of words?

What follows is a presentation of the complete network, in which morphological tags, lemmas and alignment are predicted. Model variants necessary for answering the preceding questions are discussed later in the section.

## 3.2 Factorization and Independence assumptions

Figure 3 depicts the probabilistic graphical model that represents our approach to the problem. $X$, $Y$, $A$ and $M$ are random variables as described in section 2.5. Let $L$ be an observed categorical random variable taking lemmas of Turkish words drawn from lemma vocabulary $\mathcal{V}_l$, and let $T$ and $T_{\text{hist}}$ be random variables taking morphological analyses of Turkish words drawn from the tag vocabulary $\mathcal{V}_t$.

A Turkish sentence $X_1^m$ is annotated for morphology and lemmas at the word level. $L_1^m = \langle L_1, ..., L_m \rangle$ is a random sequence of Turkish lemmas corresponding to a Turkish sentence, and $T_1^m = \langle T_1, ..., T_m \rangle$ is the corresponding random sequence of Turkish morphological analyses. Note that each $T_i$ is itself a random sequence of $o$ morphological tags, denoted by $T_i = \langle T_{i,1}, ..., T_{i,o} \rangle$. $T_{\text{hist}}$ is a random sequence like $T_1^m$, where, for a tag $t$, $t_{\text{hist}}$ is the previous tag in the sequence. For details on how morphological tags are associated with the data, see section 4.1.

The English sentence is a random sequence of $n$ English words, denoted as $Y_1^n = \langle Y_1, ..., Y_n \rangle$. Note that the English sentence is not morphologically analyzed. Finally, corresponding to each assigned word $y_j$ in this sentence we have an alignment $a_j$, which is an integer between 0 and $m$

that represents the index in the Turkish sentence for the Turkish word that generated the English word. Note that index 0 represents the null alignment, which occurs when a word isn't aligned to any word in the translated sentence.

The joint distribution implied by figure 3 is $P(L_1^m, T_1^m, Y_1^n, A_1^n | X_1^m, M, T_{\text{hist}})$. Given that $X$ is observed, we can make multiple independence assumptions, notably that $L \perp T$, $L \perp Y$ and $T \perp Y$. each of which give us a Conditional Probability Distribution (CPD). The joint distribution can be factorized as follows:

$$P(L_1^m, T_1^m, Y_1^n, A_1^n | X_1^m, M, T_{\text{hist}}) = P(T_1^m | X_1^m, T_{\text{hist}}) \times \tag{14}$$
$$P(L_1^m | X_1^m) \times \tag{15}$$
$$P(Y_1^n, A_1^n | X_1^m, M) \tag{16}$$

Each part of this product can be further described. First, the tag component of the distribution is written as follows:

$$P(T_1^m = t_1^m | X_1^m = x_1^m, T_{\text{hist}} = t_{\text{hist}}) = \prod_{i=1}^{m} P(t_i | x_i, t_{\text{hist}}) \tag{17}$$
$$= \prod_{i=1}^{m} \prod_{j=1}^{o} P(t_{i,j} | x_i, t_{\text{hist}}) \tag{18}$$
$$= \prod_{i=1}^{m} \prod_{j=1}^{o} P(t_{i,j} | x_i, t_{i,<j}) \qquad \text{Precise history access} \tag{19}$$

By precise history access, we here refer to the decision to let $t_{i,j}$ depend on $t_{i,<j}$, which will be further dealt with in the implementation section. The lemma component of the distribution becomes the following:

$$P(L_1^m = l_1^m | X_1^m = x_1^m) = \prod_{i=1}^{m} P(l_i | x_i) \tag{20}$$

Marginalization over the latent alignment variable is only relevant for the lexical component of the distribution. The marginal distribution for this component is $P(Y_1^n = y_1^n | X_1^m = x_1^m, m)$, which is necessary for training as alignment information is unavailable. It can be further rewritten as follows:

$$P(Y_1^n = y_1^n | X_1^m = x_1^m, m) = \sum_{a_1^n} P(Y_1^n = y_1^n, A_1^n = a_1^n | X_1^m = x_1^m, M = m) \quad \text{Marginalization} \tag{21}$$

$$= \sum_{a_1^n} \prod_{j=1}^{n} P(y_j, a_j | x_1^m, m) \tag{22}$$

$$= \sum_{a_1=0}^{m} ... \sum_{a_n=0}^{m} \prod_{j=1}^{n} P(y_j, a_j | x_1^m, m) \tag{23}$$

$$= \prod_{j=1}^{n} \sum_{a_j=0}^{m} P(y_j, a_j | x_1^m, m) \tag{24}$$

$$= \prod_{j=1}^{n} \sum_{a_j=0}^{m} P(a_j | x_1^m, m) P(y_j | x_1^m, m, a_j) \tag{25}$$

$$= \prod_{j=1}^{n} \sum_{a_j=0}^{m} P(a_j | m) P(y_j | x_{a_j}) \quad \text{Independence and IBM assumptions} \tag{26}$$

The following section will describe how we parameterize the resulting probability distributions in equations 26, 20 and 19.

### 3.3 Parameterization and Implementation

Having defined the model and its independence assumptions, and having arrived at several conditional probability distributions, we are now ready to describe how these distributions will be estimated. The parameterization and its implementation describes the flow of transformations from the input to predicted probabilities. We now show separately how the probability distributions $P(Y|X = x)$, $P(A|M = m)$, $P(L|X = x)$ and $P(T|X = x, T_{\text{hist}} = t_{\text{hist}})$ are parameterized.

$P(A|M = m)$ is parameterized by the uniform distribution; $\mathcal{U}(\frac{1}{m+1})$, which is the same distribution as the one used in IBM Model 1. The other distributions are all similar to each other in their form; $P(L|X = x)$ is modeled by $\text{Cat}(\sigma(f_\theta(x)))$, $P(T|X = x, T_{\text{hist}})$ is modeled by $\text{Cat}(\sigma(g_\theta(x, t_{\text{hist}})))$ and $P(Y|X = x)$ is modeled by $\text{Cat}(\sigma(h_\theta(x)))$, where Cat is the categorical distribution and $\sigma$ the a softmax function. Functions $f_\theta(X)$, $g_\theta(X)$ and $h_\theta(X)$ are complex functions that combine NLP architectures described previously. Figure 4 shows the entire architecture graphically, where the dimensions represent a single sentence passing through. In the written description below it will sometimes be simpler to treat the functions at the word level. Note that in the practical implementation a batch dimension is also necessary for the network to handle multiple sentences at the same time[5].

For the prediction of tags and lemmas, we used the type of encoded representation that we expected to be most applicable to the prediction being made. The occurrence of tags is directly related to the presence of morpheme surface forms. These should be found easily by a convolutional

[5]That is, the batch dimension will be ignored in this description; each data structure has an extra dimension in practice.

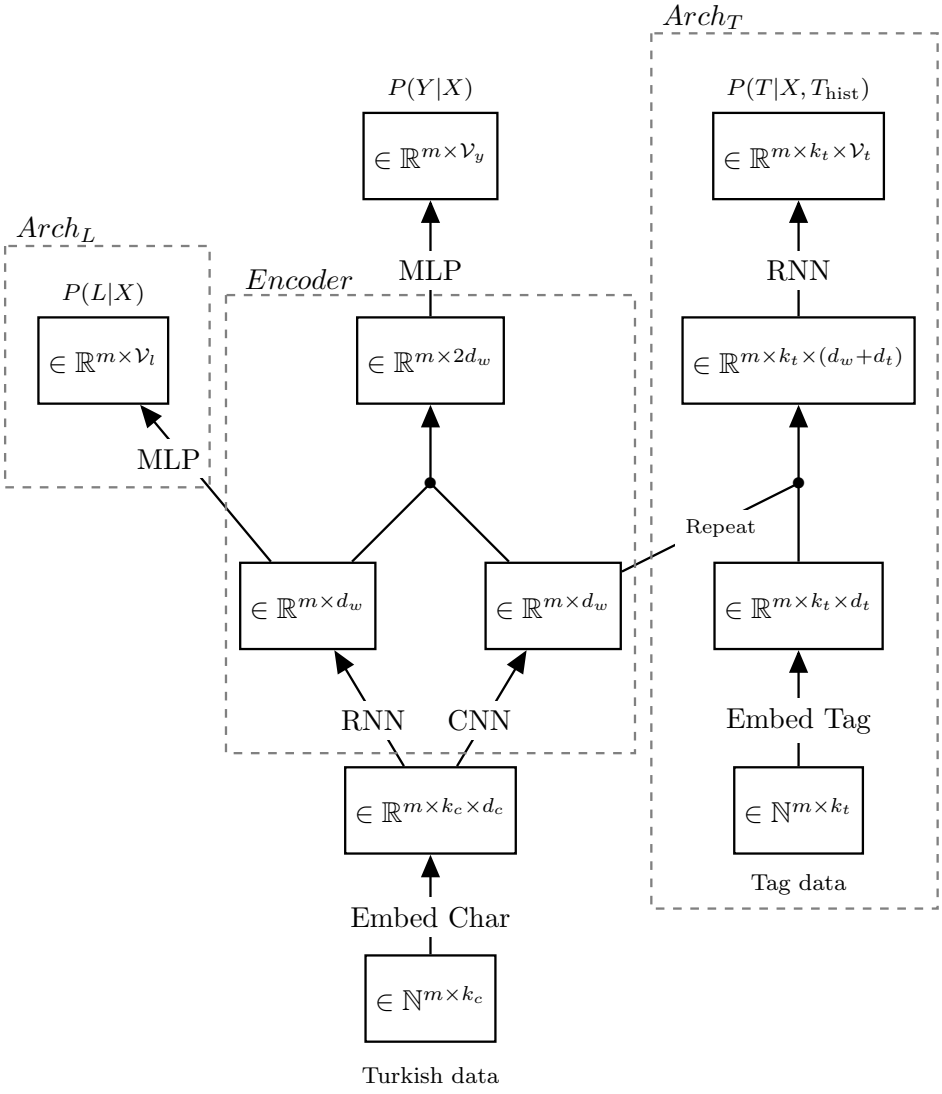



Figure 4: Model 1's architecture. Expressions in boxes denote the dimensionality of the data structure. Labels on arrows denotes what type of function transforms the data. Merging arrows represent concatenation. The CPDs above the leftmost and top two boxes show the probability distribution that the node will predict. Incoming data types are shown at the bottom of the figure. Finally, the dotted boxes represent segments of the network that will be changed for variants of the model.

encoder due to its focus on small, local features. For instance, a filter width of 3 can check for the occurrence of a morpheme constructed of 3 characters.. On the other hand, a bi-directional GRU is applicable for capturing the semantics of possibly longer sequences such as a lemma, and gives the network an opportunity to encode other types of features that are useful for predicting $P(Y|X = x)$.

Each of the functions are identical in first transforming their input $x$ into a character-level representation $\mathbf{C}^x$ by embedding the characters of each word, as described in section 2.6.1. For a single sentence, the resulting representation is a three-dimensional tensor $\in \mathbb{R}^{m \times k_c \times d_c}$.

For distribution $P(L|X = x)$, function $f$ then uses an RNN encoder for each word to arrive at a vector representation $e_{\text{RNN}} \in \mathbb{R}^{d_w}$ that can be seen as a word embedding. The RNN architecture uses bidirectional GRUs as presented previously, and returns only the final state for each word instead of a sequence. To save parameters, the outputs from the two directions are merged by summing the vectors rather than concatenating them. The output from the RNN is then used as input to an MLP, which outputs a vector $\in \mathbb{R}^{|\mathcal{V}_l|}$ per word, where $\mathcal{V}_l$ is the vocabulary of lemmas. The resulting vector is the output of function $f$, and after a softmax activation function it attempts to approximate the distribution $P(L|X = x)$, predicting the probability of each lemma in the vocabulary given its input.

Function $g$, used to model $P(T|X = x, T_{\text{hist}})$, also creates a word representation for $x$, but does so using a CNN encoder. The architecture is the same as described in section 2.6.4; for each word it takes as input a matrix $\in \mathbb{R}^{k_c \times d_c}$ and outputs a vector $e_{\text{CNN}} \in \mathbb{R}^{d_w}$, where $d_w$ can be determined by the number of filters in the CNN. This vector is used in conjunction with the previous morphological tag ($T_{i,j-1}$) to predict the next tag ($T_{i,j}$).

Previous morphological tags are supplied directly from the data for this purpose. Each tag sequence has an end-of-word tag appended, and each input tag in the sequence is then embedded in the same way as a character. For a word, this produces a matrix $\in \mathbb{R}^{k_t \times d_t}$, where $k_t$ is the length of the word (number of morphological tags) and $d_t$ is the embedding size, defined to be the same size as $d_w$.

To predict a sequence of morphological tags from these inputs we use another RNN. This time, the RNN outputs a sequence of vectors, each of which represents a distribution over tags. The input to each hidden state in the network is a vector $\in \mathbb{R}^{k_t \times d_w + d_t}$, which is the concatenation of the CNN embedding $e_{\text{CNN}}$ and the tag embedding for $T_{i,j}$.[6] The network outputs a sequence of vectors $\in \mathbb{R}^{|\mathcal{V}_t|}$, where $\mathcal{V}_t$ is the vocabulary of all tags. This is the output of function $g$, treated similarly to function $f$'s output.

Note that it would in principle also be possible to predict all tags directly from a single vector (without supplying the true tags as input). With added context, however, the task becomes significantly easier. With the true, previous tag supplied at every time step, the model's output can be seen as an upper bound on performance.

Finally, function $h$ for $P(Y|X = x)$ encodes its word embedding using both the RNN encoder and the CNN encoder. The output of both encoders, $e_{\text{RNN}}$ and $e_{\text{CNN}}$, is concatenated to produce a vector $\in \mathbb{R}^{2d_w}$ for each word. This representation is the input to a densely connected MLP, which produces a vector $\in \mathbb{R}^{|\mathcal{V}_y|}$, where $\mathcal{V}_y$ is the vocabulary of English words. As is the case for the other functions, the output of this MLP can model a probability distribution after applying a softmax activation function. The multiplication of the alignment and lexical distribution was implemented as a dot product of the data structures. This step is not included in figure 4 as it is not a part of

[6] A few technical details are worth mentioning here. First, notice that $e_{\text{RNN}}$ is used repeatedly as input to the RNN. Secondly, for predicting the first tag, $T_{i,0}$ is defined as a vector of zeroes, so that the prediction is made using only the word representation. Lastly, the end-of-word tag is only predicted, so it is excluded from input.

the architecture. It is, however, still a part of the model's graphical structure.

Between each of the functions, some of the parameters are shared. Notably, the parameters for the character embeddings are the same, and function $h$ shares the RNN encoder parameters with $f$ and the CNN encoder parameters with $g$. Backpropogation of the learning signal for one distribution can affect the modeling of another distribution because there is some overlap between the parameters, which allows the distributions to bias each other.

Note that it would have been theoretically possible to model $P(Y|X_1^m, A)$, for instance by creating contextually informed representations of the words in $X_1^m$ by passing them through a word-level RNN. $Y$ would then be conditioned not on a single word $X_a$, but on a representation $r_a(X_1^m)$ (a word vector contextually informed by its neighbors, pointed to by alignment $a$). Instead, we opted to focus on the lexical aspect, rewarding the model for pushing its disambiguation power into the isolated representation of $X_a$. This more clearly shows whether the model is able to usefully alter its lexical distribution.

The network was implemented using Keras, a high-level API for neural networks (Chollet et al., 2015). The Python library Theano, for efficient and fast handling of multi-dimensional arrays on GPUs, was used as the backend (Theano Development Team, 2016).

### 3.4 Determining Alignments and AER

Note that the architecture as presented thus far predicts a number of distributions, but does not yet return a sequence of alignments. We use a viterbi algorithm to predict the most likely alignment for each word in the English sentence, $\hat{a}_1^n$. This value is found as follows:

$$\hat{a}_1^n = \underset{a_1^n}{\text{argmax}}\, P(a_1^n|x_1^m, y_1^n) \tag{27}$$

$$= \underset{a_1^n}{\text{argmax}}\, P(a_1^n, y_1^n|x_1^m) \tag{28}$$

$$= \underset{a_1^n}{\text{argmax}} \prod_j P(a_j|m)P(y_j|x_{a_j}) \tag{29}$$

$$= \left( \underset{a_j \in \{0,\ldots,m\}}{\text{argmax}} P(a_j|m)P(y_j|x_{a_j}) \right)_{j=1}^{n} \tag{30}$$

This approach uses information from the lexical distribution to find the most likely alignment. The argmax in equation 30 considers, given a specific word in the English sentence $y$, the probability distribution over all the words in the Turkish sentence $\langle x_1, ..., x_m \rangle$, multiplied by the alignment probability. It selects the most likely Turkish word, and returns the index $\hat{a}_j$ (between 0 and $m$) associated with it. This argmax is performed for every word in the English sentence from $j = 1$ to $n$, producing the prediction $\hat{a}_1^n$. It is worth noting that this formula could be further reduced (as in equation 31) specifically for IBM Model 1, since the argmax function is not affected by the uniform alignment distribution. However, this was not implemented in practice to keep the network more easily adaptable for more complex distributions.

$$\hat{a}_1^n = \left( \underset{a_j \in \{0,\ldots,m\}}{\text{argmax}} P(y_j|x_{a_j}) \right)_{j=1}^{n} \tag{31}$$

Given these predicted alignments, we can then calculate the AER as presented previously, taking every alignment prediction as sure. The alignments are predicted for sentences in the validation and test sets, and the AER is calculated by comparing those predictions to gold-labeled alignments associated with the textual data[7].

### 3.5 Model Variants

To answer the questions stated at the start of this section, a few variants of the complete network need to be defined. The complete network will be referred to as model 1. To investigate the effects of including various types of linguistic information, we also train a model with only morphological tag data (model 2), a model with only lemma information (model 3) and a model that does not have a linguistic bias besides predicting aligment (model 4). Model 2 only predicts distributions $P(Y|X = x)$ and $P(L|X = x)$. In figure 4 this corresponds to excluding $Arch_T$ from the architecture. Model 3 only predicts $P(Y|X = x)$ and $P(T|X = x, T_{\text{hist}})$, which in figure 4 is equivalent to dropping $Arch_L$. Model 4, then, predicts only $P(Y|X = x)$, excluding both side architectures. Note that all these variants still have both an RNN encoder and a CNN encoder.

To investigate which representations best encode the input, we also present two models that change the encoder. Model 5 also predicts only $P(Y|X)$, but uses solely an RNN encoder. Model 6, conversely, uses only a CNN encoder. To provide a fair comparison between these models and model 4, the dimensionality of the word representations that are input to the final MLP is kept constant. That is, whereas the RNN and CNN output vectors each have a dimensionality of 64 before concatenation in model 4, they will have a dimensionality of 128 in models 5 and 6. In figure 4 these changes would mean replacing the entire encoder with a single arrow (either a CNN or an RNN), leading straight to a data structure $\in \mathbb{R}^{m \times 2d_w}$.

Finally, we include a baseline model that works entirely on the word-level. Model 7 uses only word embeddings as its encoded word representation, meaning that the architecture consists of just an embedding layer and an MLP. This comparison serves to elucidate the value of moving to character-level representations.

### 3.6 Related Work

Recently, other projects have attempted to work on machine translation involving morphologically complex languages from a similar angle (Vylomova et al., 2016). They present neural architectures for learning word representations built from different lower levels. A bag-of-morphs model and a bi-directional LSTM model build representations from sub-word unit input, while a CNN model inspired by Kim et al (2016) and another bi-directional LSTM model build their representations from the character-level. Their models outperform various baselines for translating morphologically complex languages.

Thus, they also explore different ways to linguistically bias the model; their approach changes form of this input. By contrast, this project biases the model by also letting the input predict probability distributions that should be relevant to the task. The indirect supervision provided could further improve results.

It should be noted that the approach presented here is different from multi-task learning. In multi-task learning, parameters are shared over a series of different, synergistic tasks (which may

[7]Since human-annotated alignment data was only available for the validation and test sets, these scores were not calculated for training.

| | Turkish | English |
|---|---|---|
| Number of sentences | 207678 | 207678 |
| Number of tokens | 4226965 | 5091986 |
| Average sentence length | 20.35 | 24.52 |
| Vocabulary size | 179007 | 74924 |
| Number of singletons | 86667 | 31347 |

Table 2: Corpus Statistics

be using entirely different sets of input data) (Goldberg, 2016). Recent examples include combining tasks on machine translation and syntactic parsing (Luong et al., 2015) and combining syntactic chunking, CCG-supertagging and POS-tagging (Søgaard and Goldberg, 2016). This model, on the other hand, models a joint probability distribution. In particular, the predictions made are statistically dependent through the joint model and the conditional factors. In this way, the predictions are more intricately linked than just through parameters that happen to be shared.

# 4 Experiment

## 4.1 Data

The corpus of parallel data was attained from the shared news translation task at WMT 2016. For the Turkish-English language pair, the corpus consists of 207678 translated sentences obtained from the Southeast European Times as part of the OPUS project (Tiedemann, 2009).

Both the corpora were tokenized using the Natural Language Toolkit tokenizer, and naively normalized by lowercasing each word. A 'beginning-of-word' token that represents the null alignment is added at the beginning of each Turkish sentence, to allow words on the English side to not align to anything if necessary[8]. No 'end-of-word' tokens were used. Some characteristics of the data after preprocessing are shown in table 2. This table exemplifies the data sparsity issues that a morphologically complex language brings; the Turkish vocabulary of this (relatively small) corpus is over twice as large as the English vocabulary. It contains almost three times as many words that only occur once in the corpus; in fact, the number of singletons on the Turkish side is significantly larger than the entire English vocabulary.

## 4.2 Validation and Test Data

For validation and test data, a gold standard alignment data set was used (Cakmak et al., 2012). The complete set consists of 300 parallel English-Turkish sentences which have been manually aligned. The sentences are high quality, exact translations, taken from language proficiency exams.

This set of 300 sentences was randomly shuffled and morphologically analyzed exactly as the training data, as described in the next section. Then 100 sentences were selected for the validation set, which was used for developing the architecture and selecting models. The other 200 sentences were used as a test set for the final 7 models.

[8]If the Turkish side does not align to anything

## 4.3 Morphological Analysis

To provide morphological tags to the network, the SETIMES corpus was analyzed using a morphological parser and disambiguator (Sak et al., 2008). The parsing tool starts from a lexicon of 54,267 lemmas, takes into account the phonological rules caused by vowel harmony, and uses a transducer to output all possible morphological analyses of a word. For instance, given the input **için**, the parser returns the possible analyses in table 3.

| |
|---|
| iç[Verb]+[Pos]+[Imp]+YHn[A2pl] |
| için[Postp]+[PCNom] |
| için[Postp]+[PCNom]+[A3sg]+[Pnon]+[Nom] |
| iç[Adj]-[Noun]+[A3sg]+Hn[P2sg]+[Nom] |
| iç[Adj]-[Noun]+[A3sg]+[Pnon]+NHn[Gen] |
| iç[Noun]+[A3sg]+Hn[P2sg]+[Nom] |
| iç[Noun]+[A3sg]+[Pnon]+NHn[Gen] |

Table 3: Analyses of **için**

In these analyses, each expression inside square brackets represents a morphological tag. These refer to many possible morphological features in Turkish, including number, case, tense, aspect and negation, with the first tag always being the POS-tag of the root. The tags are separated by either a **-** or a **+**, which indicates whether the following tag comes from a derivational (**-**) or an inflectional (**+**) morpheme. A string of characters outside the square brackets is part of the word's surface form; the first string is the lemma, while others are morphemes. Capital letters in morpheme strings are there to take into account vowel harmony; every variation of a morpheme is thus captured by a single string.

After parsing, the morphological disambiguator reranks the parser's output using a perceptron algorithm. For each parse sequence for each word in the sentence, it computes the most likely parse sequence given the sentence. For the example word **için**, the most probable parse given the sentence was predicted to be **için[Postp]+[PCNom]**[9].

From the disambiguators output, the most likely analysis of each word was taken, and this analysis was processed into a sequence of morphological features. In other words, our model for morphology ignores whether the feature is derivational or inflectional, and excludes surface forms. For this example, our model attempts to predict [[**Postp**],[**PCNom**]] from **için**. For clarity it is worth repeating here that both tools mentioned are presented by Sak et al. (2008).

This set of tools was chosen for a number of reasons. Importantly, it was well-suited for the problem since it provided the type of information we were hoping to give the model access to, such as grammatical case, person, mood and tense. Some other resources used a far coarser categorization. Practical considerations also informed the choice: the analyzer was entirely automated, so that no manual intervention was necessary, it was relatively easy to implement and it was available as open-source software. Finally, the authors report 98% accuracy on the disambiguation task, so the input to the network is mostly valuable. This does not take into account the accuracy of the parser, however. For most Turkish words, the parser simply lists all possible parses, but if it encounters

[9]This word comes from the Turkish sentence "bazı durumlarda şirketler , onları işletme isteğinden çok , üstünde oturdukları arsalar için satın alınıyor . ", which can be translated as "in some cases , companies were bought more for the land than for any desire to keep operating them . ". In this context 'için' means 'for', so that the analysis as a postposition is justifiable.

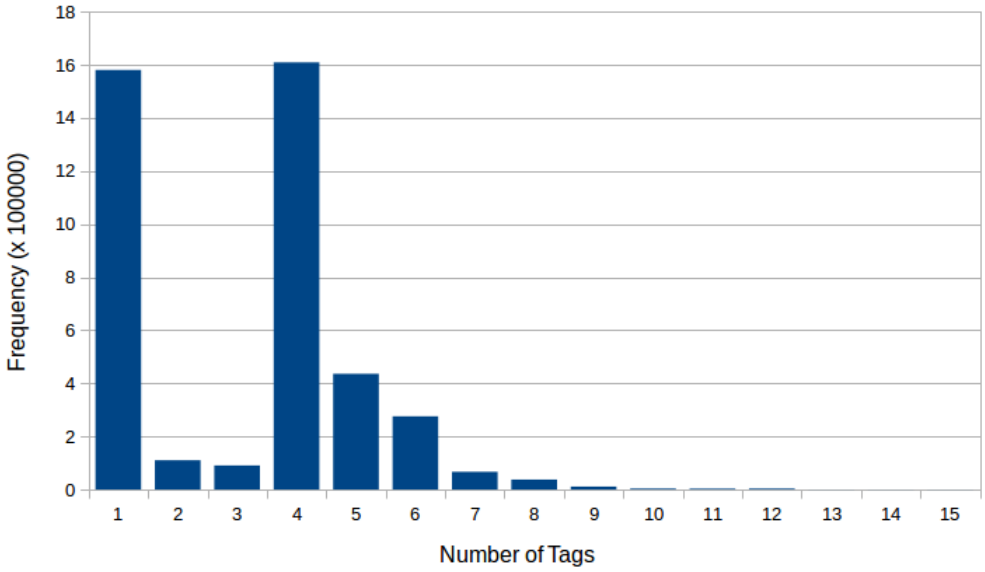



Figure 5: Histogram for Number of Tags per Word

a word with no valid parse it outputs the tag '[Unknown]'. This happens often for proper nouns, foreign words and other unexpected inputs, so that information is lost.

Because the tool's output is an approximation, using it inevitably brings noise into the data. This may be exacerbated by the fact that the corpus which it analyzes differs from the data on which high accuracy scores were reported, so that the actual accuracy might be lower. Training a network on data that is not entirely gold-label will almost certainly limit performance. Unfortunately, finding the approximate accuracy of the tagger on the SETIMES dataset was beyond the scope of this project.

Some statistics regarding the data set may be insightful here. Figure 4.3 shows the distribution of number of tags per word. Two high peaks occur for words with 1 or 4 tags, after which the tail of the distribution drops quickly. The peak at 4 seems to be caused by simpler words often being analyzed using 4 features[10]. That is, simple words with a low number of features normally get either 1 or 4 tags, and anything in between should be regarded more as an exception. In the data set there is a single word that has 15 morphological tags attached to it. Note that about 25% of the single morpheme words (a total of around $400,000$ tokens) were analyzed as unknown, which skews the distribution. In total there are 112 different tags available.

## 4.4 Training Details

Stochastic gradient descent was performed using the Adam algorithm (Kingma and Ba, 2014). The default initial learning rate (0.001) was used, because it performed best in early experiments. A

[10]For instance, nouns were often analyzed with a tag for the root POS-tag, number agreement, another number agreement and finally a case marker

more aggressive initial learning rate of 0.01 led to unstable convergence, while a passive learning rate of 0.0001 converged to similar values as the default, taking about three times as many epochs. The Adadelta optimizer was also considered, but convergence was similar to passive Adam learning rates, and did not lead to better results than the Adam optimizer. Convergence criteria were based first on loss on the validation set, and then AER for the validation set. A batch size of 64 was used, as larger batch sizes caused memory issues.

In order to allow the network to predict probability distributions, labeled data was transformed into a one-hot encoding. For instance, this converts an English sentence $\in \mathbb{R}^m$ into a matrix $\in \mathbb{R}^{m \times \mathcal{V}_y}$, where each element is either a 1 or a 0. This can be thought of as a categorical probability distribution peaking at a single point. Outputs from the network can be compared to this distribution, and the network attempts to learn to let its distribution peak at the same position as the labeled data.

Word representations were chosen to be a dimensionality of 128, which is in the typical range for NLP problems. A higher embedding size (say, 256) could perhaps have performed better, but this was not feasible due to limitations in available memory. Thus the number of output units for the RNN encoder and the number of filters for the CNN encoder were set to 128. Morpheme and lemma embeddings sizes were also set to 128.

The character embedding dimension was set to 32. This parameter was tuned in the architecture without linguistic prediction but including both encoders. The network performed comparably with embedding sizes of 32 and 128 (leading to AER scores of 42.64 and 42.25, respectively). The value of 32 was chosen since it comfortably satisfied the condition $d_c < \mathcal{V}_c$, as recommended by Kim et al. (2016). Graphs of the models' performance through the epochs can be found in the appendix.

Similarly, the word embedding dimension for model 7 was tuned to be 128. The model with this dimension attained the lowest AER on the validation set, at 47.99. Graphs for tuning this hyperparameter can also be found in the appendix.

Filter widths were based on size of morpheme surface forms found in the data. As can be seen in figure 4.4, by far the majority of all morphemes were 2, 3 or 4 characters long. To bias the network to construct features corresponding to those morphemes, filter widths of those sizes seemed most applicable. Furthermore, the relative proportion of filters for each width was matched to the relative proportion of morpheme surface form lengths found in the data. Excluding other lengths, the proportion of morphemes with lengths 2, 3 and 4 was respectively 58.6%, 36.5% and 4.9%. For an embedding dimension of 128, this resulted in respectively 75, 47, and 6 filters.

Due to limited resources, some of the network's hyperparameters could not be tuned. For those hyperparameters, typical values were selected from the literature. The vocabulary size for English was set to $30,000$, and the maximum allowed sentence length was set to 40. The activation function in the networks's MLPs was ReLU, chosen for reliable perfomance and fast gradient calculation. For the CNN encoder, however, early experiments showed that the tanh function performed slightly better than ReLU. tanh was therefore used as the feature maps' activation function.

# 5 Results and Analysis

Table 4 shows results on the validation set for the best performing version of each model. Model selection was performed by choosing the version of the model with the best AER on the validation set; the model is parameterized by the parameters at the end of the training epoch for which the lowest AER occurred.

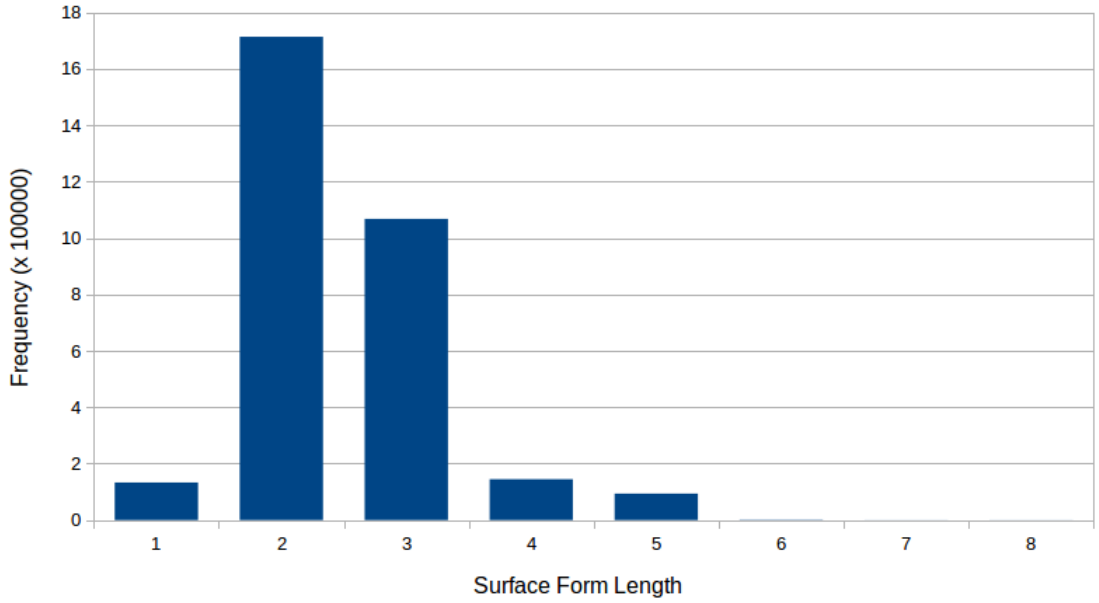



Figure 6: Histogram for Length of Morpheme Surface Forms

Table 4: Validation Results

| Model | Linguistic Supervision | Encoder | Best AER | Epoch | Total | % accuracy for $P(Y\|X)$ | $P(T\|X)$ | $P(L\|X)$ |
|---|---|---|---|---|---|---|---|---|
| 1 | Morph + Lem | RNN+CNN | 47.61 | 15 | 23 | 6.65 | 76.63 | **89.62** |
| 2 | Lem | RNN+CNN | 42.20 | 15 | 23 | 6.67 | n/a | 89.59 |
| 3 | Morph | RNN+CNN | 50.47 | 11 | 26 | 7.14 | **78.94** | n/a |
| 4 | None | RNN+CNN | 43.06 | 16 | 26 | 6.96 | n/a | n/a |
| 5 | None | RNN | **42.12** | 60 | 77 | 6.64 | n/a | n/a |
| 6 | None | CNN | 50.80 | 35 | 36 | 6.39 | n/a | n/a |
| 7 | None | Word-level | 48.07 | 20 | 23 | **9.43** | n/a | n/a |

AER on the test set was calculated for the selected models; the test results are presented in table 5. Of all models the strongest performance came from the architecture with only lemma supervision (model 2), with an AER of 42.32. Compared to other supervision models this error rate was far lower; including morphology prediction had a detrimental effect on performance, with a jump of at least 6 points. Predicting both morphology and lemmas was still slightly more productive than predicting only morphology[11].

Attaining a low AER score is possible without the added linguistic prediction; models 4 and 5 had scores that were only marginally worse than model 2. Combining both an RNN and a CNN encoder led to almost identical performance as using only an RNN. The combined encoder model was far faster to train, however, taking a fourth as many epochs to find the strongest version of the model as the RNN encoder model, and taking about a third as many epochs to converge. However, low AER scores were only attainable provided the encoder worked well; model 6, using only a CNN encoder, performed far worse. Possible improvements to the CNN encoder will be proposed in the discussion.

Moving from the word-level to a character-level encoder proved to have a beneficial effect, provided an effective encoder and effective supervision were used. The word-level model was easily outperformed by models 2, 4 and 5.

Models 1 up to and including 4 all took between approximately 17 and 20 hours to train. The CNN and Word-level models took less time to train; respectively 10 and 5 hours. Model 5 took many more epochs to converge than the rest of the models, and its training time was therefore substantially higher at 55 hours. The scores of all models throughout training, for AER and accuracy on the probability distributions, can be found in the appendix. Note that all scores presented there are on the validation set[12].

Table 5: Test Results

| Model | Linguistic Supervision | Encoder | Test AER |
|---|---|---|---|
| 1 | Morph + Lem | RNN+CNN | 48.46 |
| 2 | Lem | RNN+CNN | **42.32** |
| 3 | Morph | RNN+CNN | 50.35 |
| 4 | None | RNN+CNN | 42.70 |
| 5 | None | RNN | 42.66 |
| 6 | None | CNN | 48.60 |
| 7 | None | Word-level | 46.61 |

Although the focus is on AER, it is also worth considering the accuracy results for the three predicted distributions in the validation set. This distribution over lemmas is most accurately predicted by far, with both model 1 and 2 achieving close to 90% accuracy. For $P(T|X)$, model 3 scores close to 80% accuracy, while the model predicting both lemmas and tags scores lower. It may be that a tradeoff happens here between AER and the tag distribution. Finally, the lexical distribution is modeled best by the word-level model, while model 3 performs best out of the

[11]For comparison to the literature, Çakmat et al. (2012) achieve an AER of 40.5 with their unit based model on a small data set. However, the current project's goal is not to attain state-of-the-art results. Rather, we hope to show that our model adaptations have a positive influence, so that more complex models (that achieve higher scores) might also benefit.

[12]Also, accuracy and AER scores are presented on a scale of 0 to 1.

character-level models.

The scores for $P(Y|X)$ are significantly lower than for other distributions, which may be due to a number of factors. Firstly, the problem is significantly harder in that the correct class needs to be chosen out of a large vocabulary. Secondly, restrictions such as IBM 1 assumptions and unavailability of context make modeling this distribution hard: a translation problem such as this one has been shown to require more complex models and architectures (e.g. (Bahdanau et al., 2014)). Specifically, the distribution would benefit from being of the form $P(Y|Y_{\text{history}}, X_1^m)$. That is, each choice for a word is informed by the other words predicted in the target sentence, and each prediction has access to the entire source sentence.

# 6 Discussion

Taking into consideration model 2's relatively strong performance, it seems that there are benefits to predicting other linguistic distributions. When the model was forced to focus not only on the lexical distribution, but also on predicting lemmas, the model's ability to predict alignments improved, if only slightly. This suggests that there is some merit to giving models an inductive bias in this manner.

There is definitely caution needed in including such information, however. Not all linguistic supervision helps, as shown by the decline in performance by our architecture when predicting morphological tags. This result runs contrary to our hypothesis that some intuition regarding morphological features should aid performance. While it is possible that morphological features cannot aid alignment at all, the drop in performance can also be explained by deficiencies in the data and the encoder.

The morphological feature data may have been distractive for two main reasons. First of all, since we know that the data used as supervision is not completely accurate, it may be necessary to change the way the distribution is modeled. Specifically, the model may require some way of modeling noise, which is currently unavailable. A possible solution to this problem is to alter the architecture so it includes a VAE (Kingma and Welling, 2013). This technique would introduce an approximated Gaussian distribution, from which we sample a latent variable. This is beneficial for two main reasons. Firsly, it allows the independent distributions to now be dependent even though $X$ is observed, which allows the distributions to more effectively share information. Secondly, the random sampling procedure gives the model better tools to deal with noise, so that it is better able to learn from an imperfect data set like the one used for morphological tags.

Another issue was that the data was not always informative to the problem at hand. It was originally decided to keep the output of the morphological disambiguator as raw as possible, but some filtering may have been appropriate. Some morphological features are likely to be useful; for example, predicting the occurrence of the genitive case in a Turkish word might help align it to the English word 'of'. On the other hand, this particular data set often had words annotated with tags that did not correspond to any English surface form. Take 'altında', which should be translated as (and aligned only to) 'under' in English. The analyzed tags for this instance of the word are '[[Noun],[A3sg],[P3sg],[Loc]]', which is a lot of information given that the lemma alone could have triggered the correct alignment to the English surface form. This problem is exacerbated by the way the analyzer handles words which change their part-of-speech. In such cases, both the original tag and the new part-of-speech tag are included in the analysis, which is not likely to be useful for alignment.

Ultimately, the tagset's richness may have been counterproductive, causing the neural architecture to use its energy predicting features that are not relevant for the task. Future projects might need to find ways to morphologically analyze data in a way that produces a more consistently relevant tagset, or postprocess morphological analyses to contain only features that may correspond directly to a surface form on the target side. An alternative solution is to let the network predict the occurrence of surface level morphemes in Turkish, giving similar morphological insight while being a more concise distribution to predict.

The inability of morphological tag data to be valuable may also have been caused by an insufficiently engineered CNN encoder. As reflected in the weak performance of model 6 (using only the CNN), it may have been necessary to alter the encoder to better capture the data. For instance, feeding the word embeddings through a highway network could have led to better results, allowing the model to consider interactions between the character n-grams. According to Kim et al (2016), the basic model performs well on its own, but future projects could consider using this added network as they suggest. Due to time constraints this layer was not included.

Another valuable extension could be to use multiple layers of convolutions, in contrast to Kim et al's proposed network of a single layer. Adding layers would allow multiple groups of characters to be combined into more abstract features, which could be beneficial. Perhaps this would allow the network to better deal with Turkish vowel harmony, allowing it to relate different short surface forms with identical meanings. A further improvement to both the RNN and the CNN encoders would be allowing layers to use dropout, which has been shown to benefit performance in many types of networks as it prevents overfitting (Srivastava et al., 2014).

An improvement that could have benefited both encoders would be to build RNNs over the word-level for the prediction of the tag and lemma distribution[13]. This has been shown to be effective in many NLP applications, since it creates a word embedding that is informed of the context surrounding it. Words can have greatly different meanings depending on their context, and an architecture that acknowledges this would likely model those two distributions more accurately. This extension was not explored due to resource restrictions. All of the previously mentioned improvements could allow the CNN embedding to better handle the combination of the morphological tag distributions and the lexical distribution, which could lead to better alignment scores.

Of course, such changes would have effects on models 4, 5 and 6 as well. Some of the improvements to the CNN encoder could allow model 6 to perform more comparably to the other models. Furthermore, model 4 with the combined encoder could then be superior to model 5 not only in training time, but also in terms of AER. Overall, results from the encoder experiments suggest that using a mixed encoder as in model 4 is an effective way to make the model flexible, although this statement could be more confidently made with more work on the CNN.

It should be mentioned that engineering the encoders could be more beneficial to models 4, 5 and 6 than to 1, 2 and 3, changing claims made regarding the usefulness of linguistic supervision. Future research will have to further disentangle these questions.

It is also worth noting that performance is limited by the strong assumptions attached to IBM Model 1. The uniform alignment distribution is perhaps suitable to languages that can have very different word orders, such Turkish and English, but allowing the model to use other distributions makes it more versatile. The assumption that each English word is aligned to only a single Turkish word is also limiting to performance. This effect is perhaps less harmful for Turkish to English than the other way around; in Turkish to English one source word is more likely to generate multiple target words than the other way around. Given this assumption, the best attainable AER on

[13]The lexical distribution could be kept without context as discussed at the end of section 3.3.

the validation and test sets combined was calculated to be 3.7. As such, there are cases where a single English word is aligned to multiple words on the Turkish side, and this fact somewhat hurts performance.

Aside from considering VAEs, another interesting line of research might be to turn the problem around, using English as the source side and attempting to predict alignments to the Turkish target side. A challenging and valuable component of this would be to generate Turkish target words, for which issues of data sparsity would be far greater than for English generation. Being able to predict the morphological features that correspond to parts of the Turkish surface form would be vital to generation. In fact VAEs could be beneficial to such a project as well. Finally, it would be valuable to test the ideas proposed here on other tasks. For instance, more word-internal knowledge could lead to gains in neural machine translation.

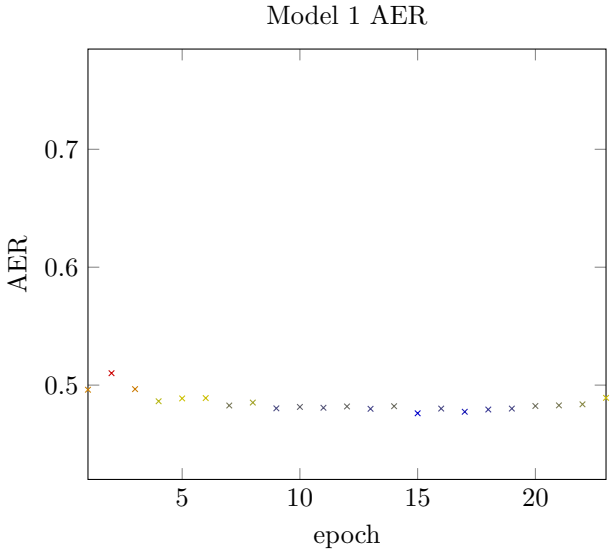
# 7 Conclusion

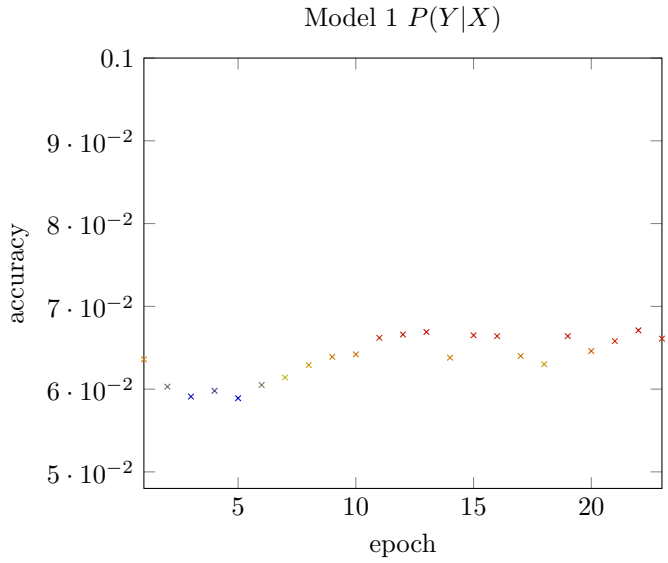
This project presented a neural architecture for predicting interrelated probability distributions to benefit word alignment performance. While the distribution over morphological features did not help performance, it was still shown that giving the model some sort of explicit linguistic bias can help alignment, as shown by model 2's strong performance. In order for this to hold, it seems pertinent that the encoder, the data and the type of linguistic feature are well chosen. Although the difference in scores was marginal for model 2, the various improvements suggested could well make the benefits more significant.

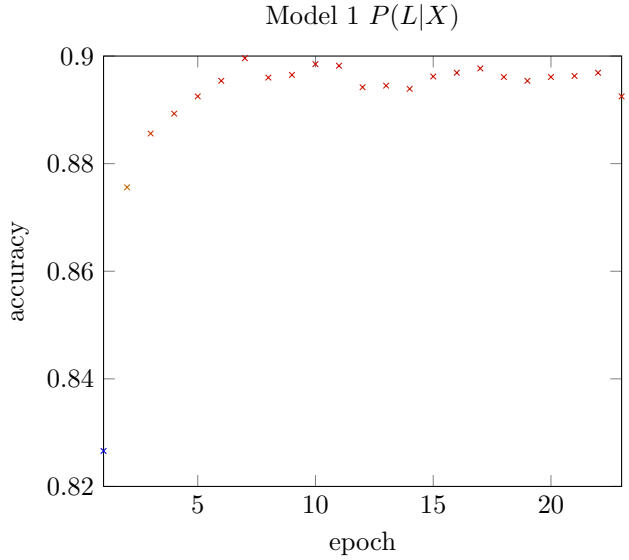
Furthermore, this project has shown that there can be merit in combining different encoders. A combined encoder performed similarly to an encoder consisting solely of an RNN, and was much easier to train, with improvements suggested that could further help the combined encoder. We have also added to the growing body of work that supports character-level models as the stronger alternative to word-level models. Finally, a number of possible directions to continue this line of research were proposed.

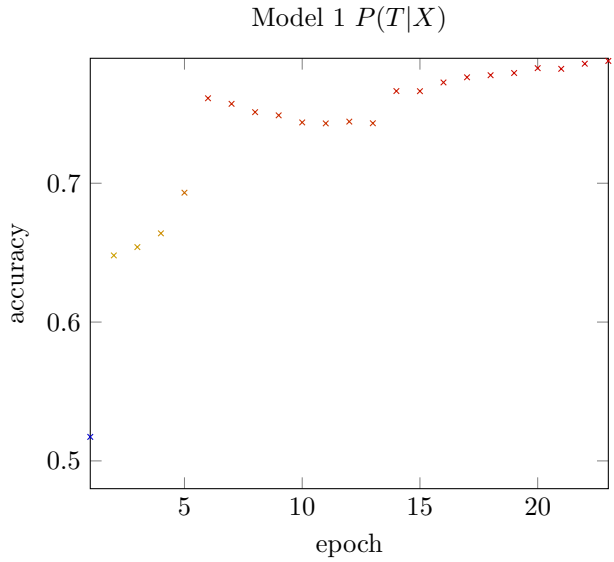
# 8 Appendix

## 8.1 Model 1 Results

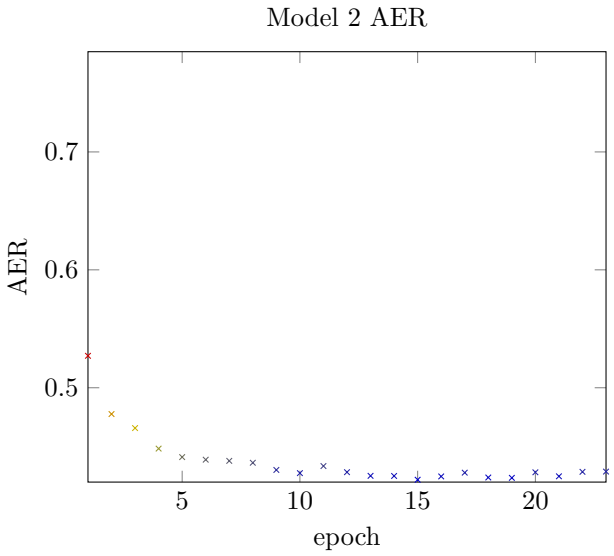


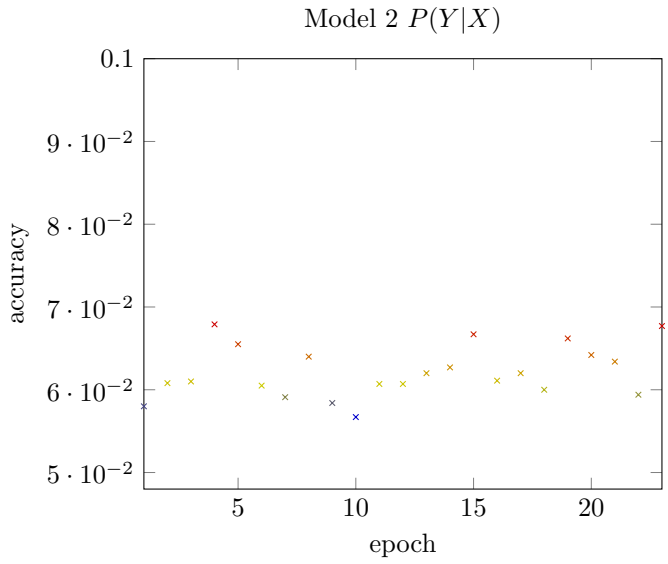



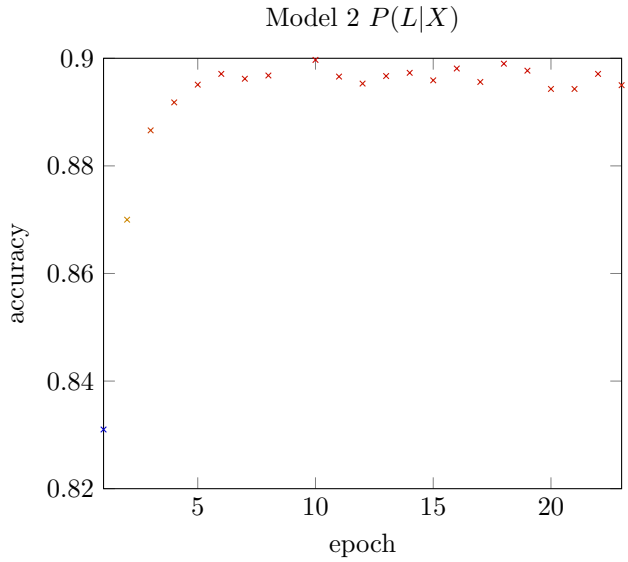





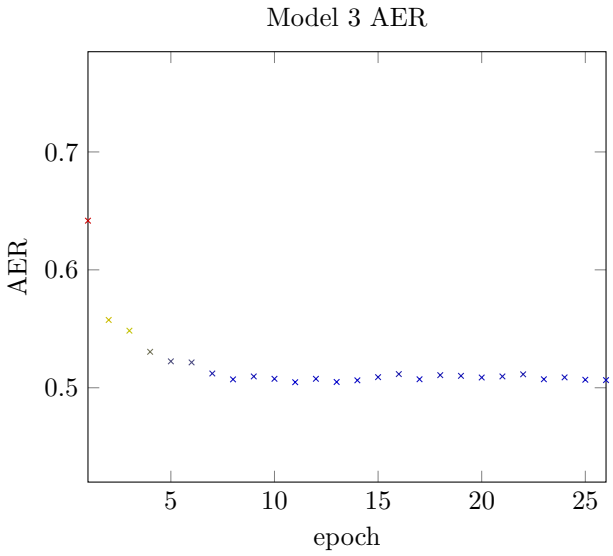

## 8.2 Model 2 Results

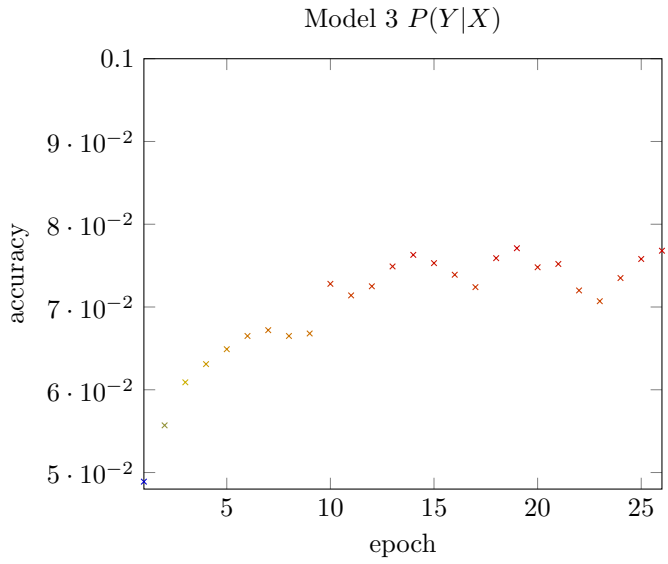


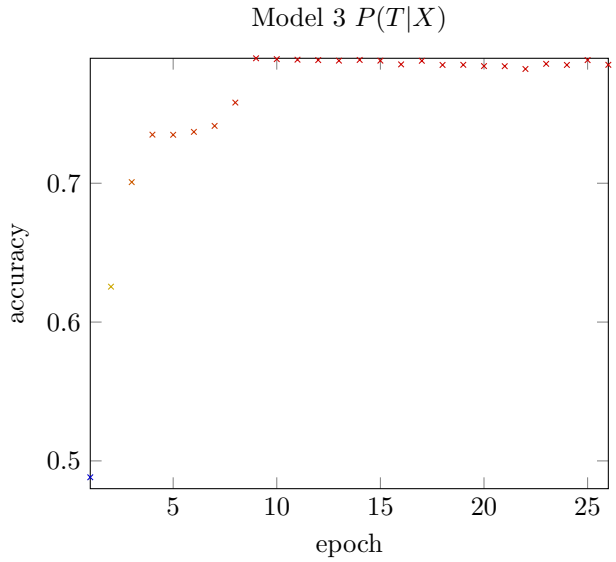





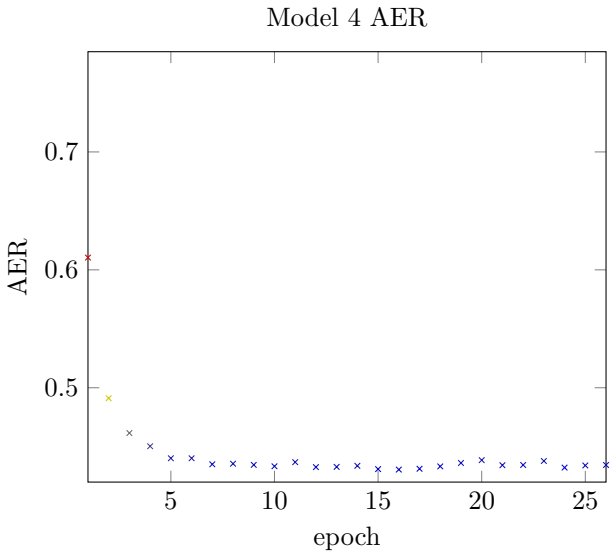

## 8.3 Model 3 Results

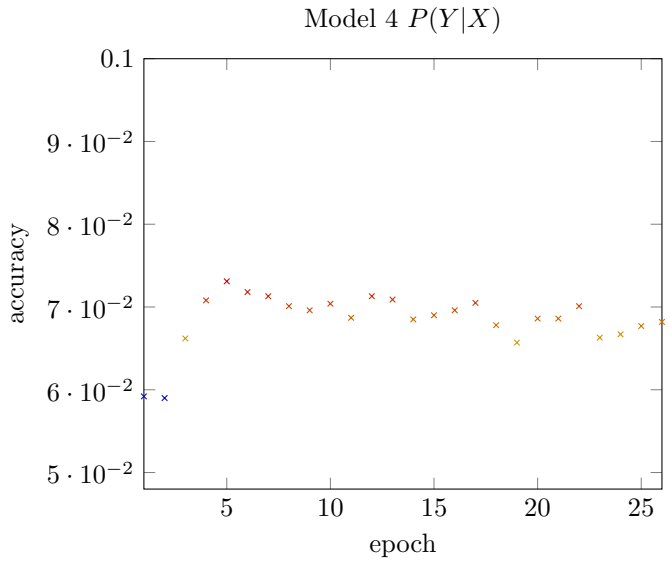


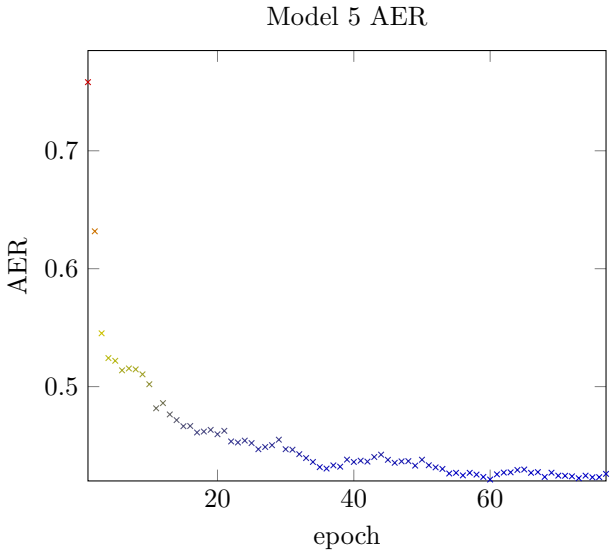


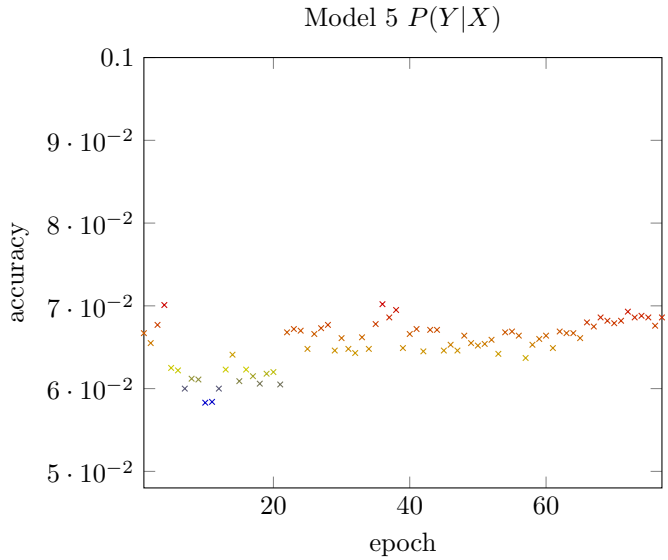

## 8.4 Model 4 Results



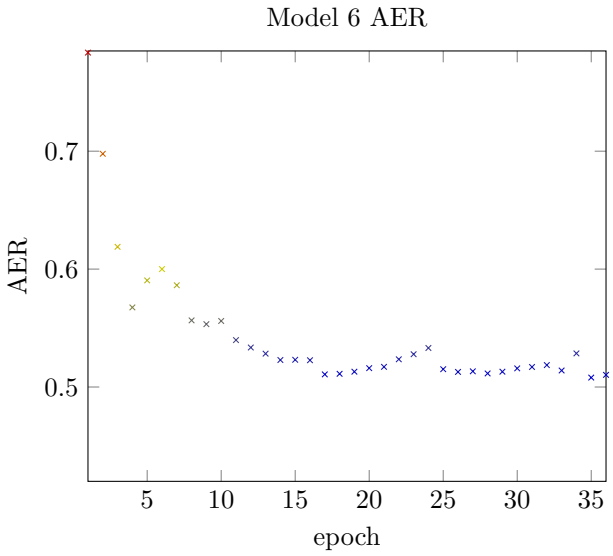



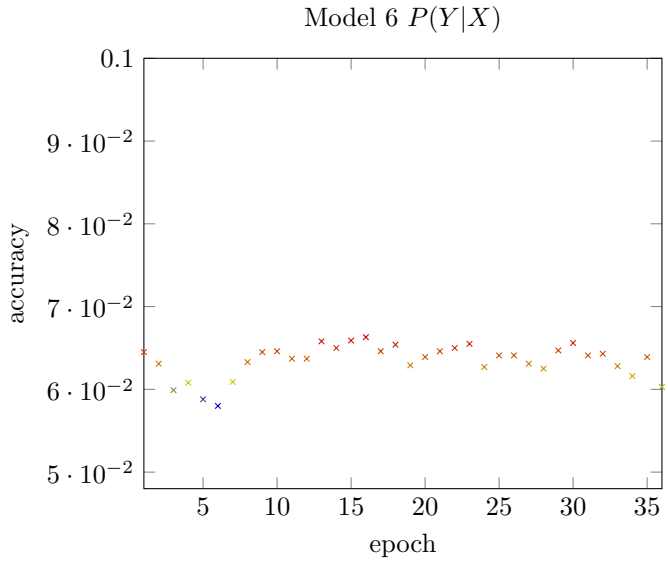
## 8.5 Model 5 Results



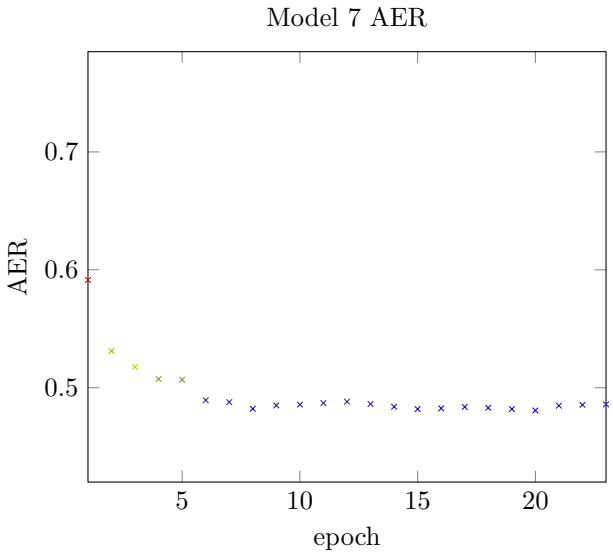



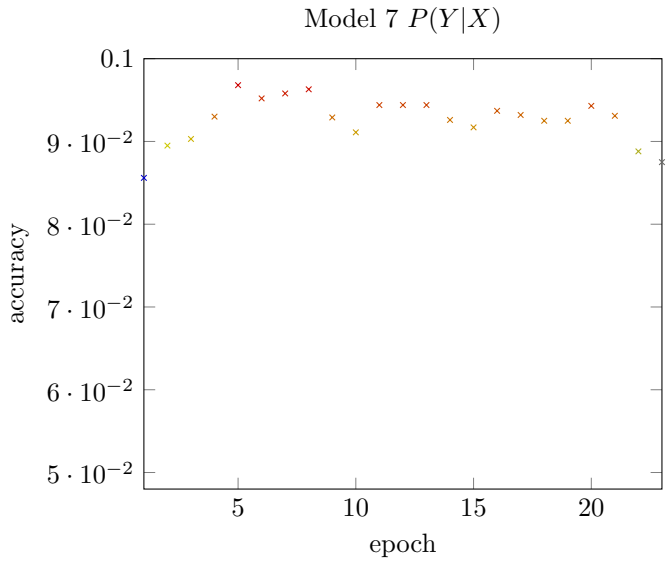

## 8.6 Model 6 Results





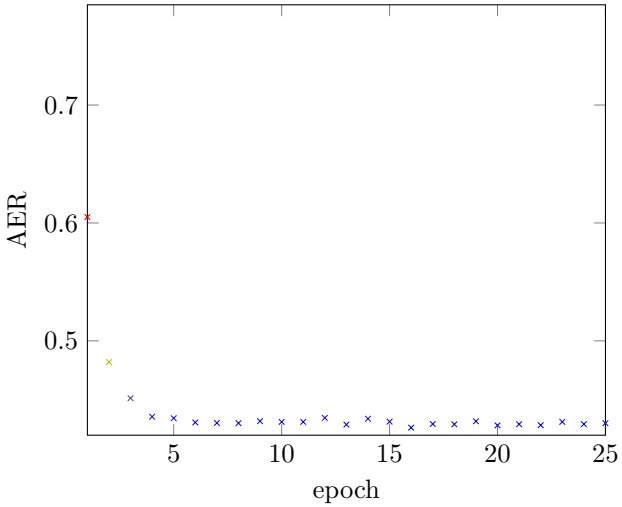

## 8.7 Model 7 Results



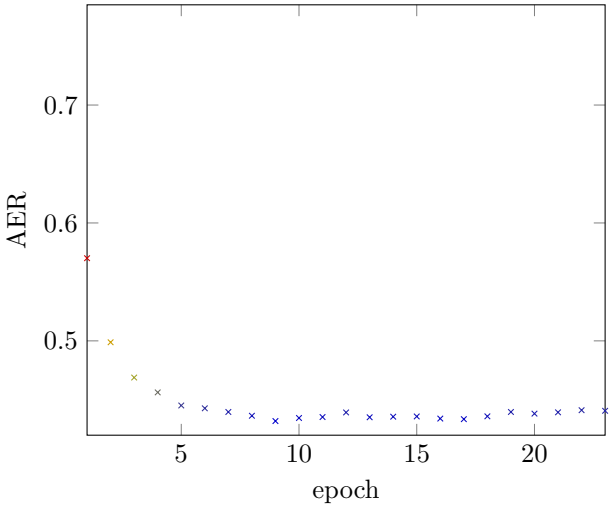



## 8.8 Tuning Character Embedding Size

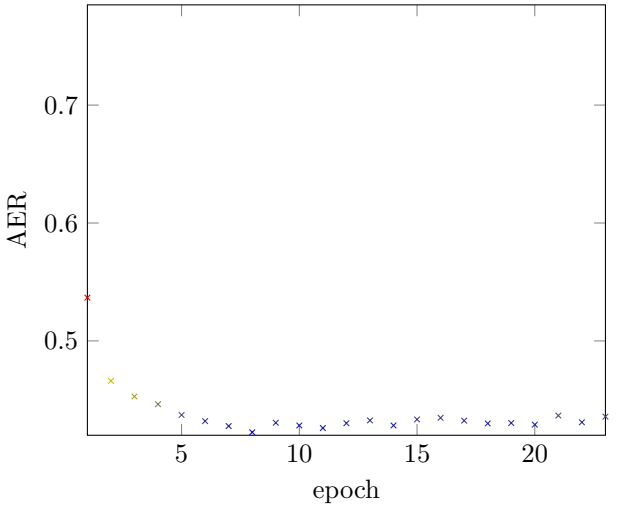
AER for Character Embedding Size 32




AER for Character Embedding Size 64

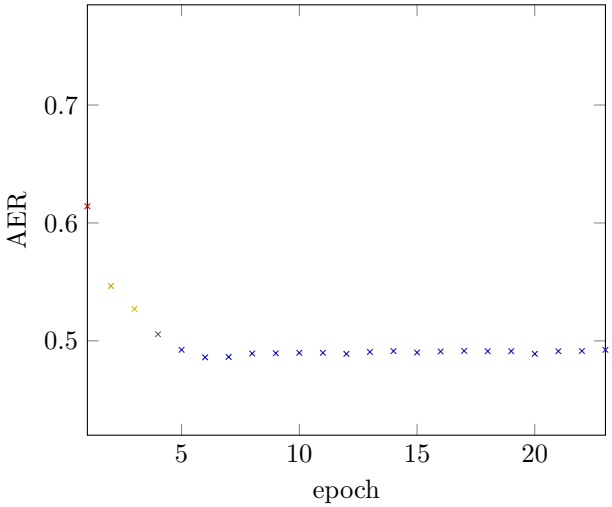



AER for Character Embedding Size 128

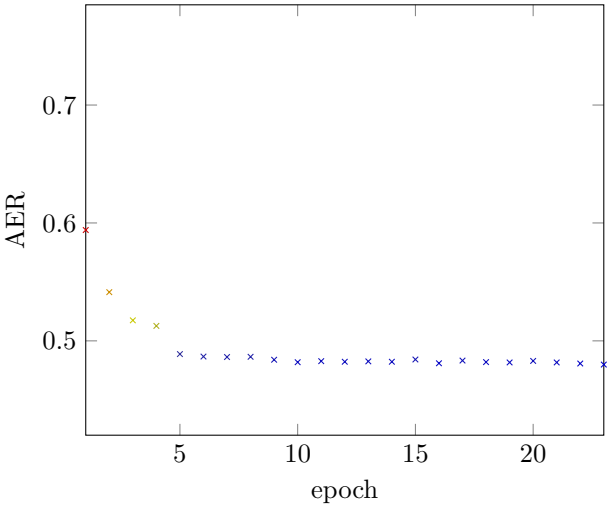


## 8.9 Tuning Word Embedding Size

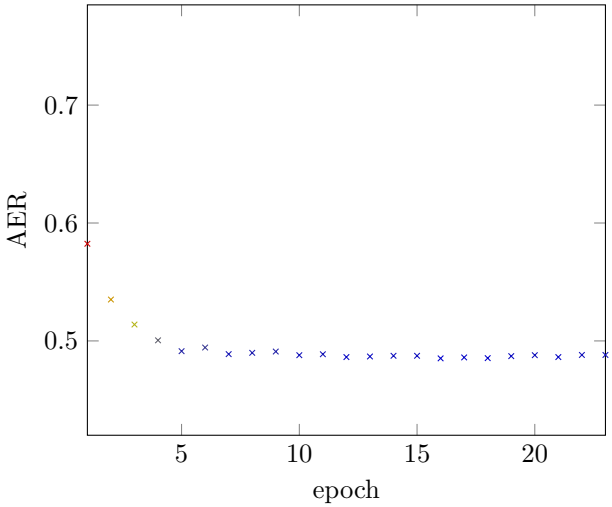
AER for Word Embedding Size 64




AER for Word Embedding Size 128




AER for Word Embedding Size 256



# References


Aikhenvald, A. Y. (2007). Typological distinctions in word-formation. In Shopen, T., editor, *Language typology and syntactic description*, volume 3, pages 1–65. Cambridge University Press.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bisazza, A. and Federico, M. (2009). Morphological pre-processing for Turkish to English statistical machine translation. In *Proceedings of IWSLT 2009*.

Brown, P., Cocke, J., Pietra, S. D., Pietra, V. D., Jelinek, F., Mercer, R., and Roossin, P. (1988). A statistical approach to language translation. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*, pages 71–76. Association for Computational Linguistics.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Cakmak, M. T., Acar, S., and Eryiğit, G. (2012). Word alignment for English-Turkish language pair. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, Istanbul, Turkey.

Carstairs-McCarthy, A. (2002). *An introduction to English morphology*. Edinburgh University Press Edinburgh.

Chiang, D. (2005). A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2016). A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.(JAIR)*, 57:345–420.

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

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2016). Character-aware neural language models. In *AAAI*, pages 2741–2749.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. In *The International Conference on Learning Representations*.

Konig, E. and Van der Auwera, J. (2013). *The Germanic languages*. Routledge.

Kuribayashi, Y. (2013). Transitivity in Turkish—a study of valence orientation—. *Asian and African Languages and Linguistics*, 7:39–52.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

Lee, J., Cho, K., and Hofmann, T. (2016). Fully character-level neural machine translation without explicit segmentation. *arXiv preprint arXiv:1610.03017*.

Lewis, G. L. (1970). Turkish grammar.

Ling, W., Luís, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., Black, A. W., and Trancoso, I. (2015). Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. (2015). Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

Oflazer, K. (1994). Two-level description of Turkish morphology. *Literary and linguistic computing*, 9(2):137–148.

Padó, S. and Lapata, M. (2009). Cross-lingual annotation projection for semantic roles. *Journal of Artificial Intelligence Research*, 36(1):307–340.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

Sak, H., Güngör, T., and Saraçlar, M. (2008). Turkish language resources: Morphological parser, morphological disambiguator and web corpus. *Advances in natural language processing*, pages 417–427.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Smith, N. A. (2011). Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274.

Søgaard, A. and Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 2, pages 231–235.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Tiedemann, J. (2009). News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In Nicolov, N., Bontcheva, K., Angelova, G., and Mitkov, R., editors, *Recent Advances in Natural Language Processing*, volume V, pages 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria.

Toutanova, K., Suzuki, H., and Ruopp, A. (2008). Applying morphology generation models to machine translation. In *ACL*, pages 514–522.

Vikner, C. and Vikner, S. (2008). Hierarchical morphological structure and ambiguity. *L'énonciation dans tous ses états-Mélanges offerts à Henning Nølke, Peter Lang Verlag, Berne*, pages 541–560.

Vogel, S., Ney, H., and Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Vylomova, E., Cohn, T., He, X., and Haffari, G. (2016). Word representation models for morphologically rich languages in neural machine translation. *arXiv preprint arXiv:1606.04217*.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657.