POINTER-GENERATOR TRANSFORMERS FOR MORPHOLOGICAL INFLECTION

A Thesis Submitted in Partial Fulfillment of

the Requirements for

the Degree of

MASTERS OF SCIENCE

at

NEW YORK UNIVERSITY

by

Assaf Singer

May 2020

POINTER-GENERATOR TRANSFORMERS FOR MORPHOLOGICAL INFLECTION

A Thesis Submitted in Partial Fulfillment of

the Requirements for

the Degree of

MASTERS OF SCIENCE

at

NEW YORK UNIVERSITY

by

Assaf Singer

May 2020

Kyunghyun Cho
Professor of Computer Science

Date

He He
Professor of Computer Science

Date

Acknowledgements

First and foremost, my highest gratitude goes to my great supervisors, Katharina Kann and Kyunghyun Cho. To Katharina, for your endless support and very wise council. Thank you for leading this research so professionally and for guiding me at times of struggle. And to Kyunghyun, for providing me the opportunity to conduct this research and explore the fascinating worlds of natural language and machine learning.

Finally, I want to thank my amazing family, that supported me when I was across the street, and when I was 6,000 miles away. Thank you for being an inspiration and a role model to me.

Assaf Singer

May 2020

Abstract

In morphologically rich languages, a word's surface form reflects syntactic and semantic properties such as gender, tense or number. For example, most English nouns have both singular and plural forms (e.g., *robot*/*robots*, *process*/*processes*), which are known as the inflected forms of the noun. The vocabularies of morphologically rich languages, e.g., German or Spanish, are larger than those of morphologically poor languages, e.g., Chinese, if every surface form is considered an independent token. This motivates the development of models that can deal with inflections by either analyzing or generating them and, thus, alleviate the sparsity problem.

This thesis presents approaches to generate morphological inflections. We cast morphological inflection as a sequence-to-sequence problem and apply different versions of the transformer, a state-of-the art deep learning model, to the task. However, for many languages, the availability of morphological lexicons, and, thus, training data for the task, is a big challenge. In our work, we explore different ways to overcome this: 1. We propose a pointer-generator transformer model to allow easy copying of input characters, which is known improve performance of neural models in the low-resource setting. 2. We implement a system for the task of unsupervised morphological paradigm completion, where systems produce inflections from raw text alone, without relying on morphological information. 3. We explore multitask training and data hallucination pretraining, two methods which yield more training examples.

With our formulated models and data augmentation methods, we participate in the SIGMORPHON 2020 shared task, and describe the NYU–CUBoulder systems for Task 0 on typologically diverse morphological inflection and Task 2 on unsu-

pervised morphological paradigm completion. Finally, we design a low-resource experiment to show the effectiveness of our proposed approaches for low-resource languages.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Morphology

In linguistics, morphology is the study of words, how they are formed, and their relationship to other words in the same language (Anderson, 2016). Morphology is concerned with analyzing the structure of words and morphemes. Many words can be related to other words by rules that are described for that language. For example, English speakers recognize that the words *dog* and *dogs* are closely related, differentiated only by the plural morpheme "-s", only found bound to noun phrases. Speakers of English recognize these relations from their innate knowledge of word formation rules. They infer intuitively that *dog* is to *dogs* as *cat* is to *cats*; and, in a similar fashion, that *dog* is to *dog walker* as *dish* is to *dishwasher*. The rules understood by a speaker reflect specific patterns or regularities in language, and in how morphemes interact.

Morphological rules can be identified by three key processes, inflection, derivation and compounding, corresponding to word change and word formation, respec-

tively. Inflection modifies a word only to express different grammatical categories such as tense, case, number and gender, while usually tending to keep the basic meaning of the word. The base form of a word is called its lemma, and other forms of a lemma are referred to as inflected forms. The set of all inflectional forms of a given lemma are called its paradigm. For example, in English, *run*, *runs*, *ran* and *running* are forms of the same paradigm. Each inflection in the paradigm expresses certain morphosyntactic features, e.g., the PAST form of *run* is *ran*. Unlike inflection which expresses only minor semantic changes in the original word, derivation is the process of forming a new word from an existing word. In many languages, this is done by adding a prefix or suffix, such as un- or -ness. For example, unhappy and happiness are both derived from the root word happy. Generally speaking, inflection applies in more consistent patterns to all members of a paradigm, while derivation follows less consistent patterns.

Compounding occurs when two or more words or signs are joined to make one longer word. In English, a word such as *blackbird* is a compound, composed of the adjective *black* and the noun *bird*.

1.2 Aim and Scope

As we fill further elaborate in subsequent chapters, handling morphology is essential for many natural language processing (NLP) tasks, especially when dealing with morphologically rich languages (Minkov et al., 2007). Therefore, in this thesis, we aim to take a closer look at morphology, with a focus on computational approaches to inflectional morphology. In recent years, major advances on deep learning research, a sub-field of machine learning, have lead to the development of

many high-performing deep learning-based natural language processing systems. These models have advanced the state of the art in various areas, such as machine translation (Cho et al., 2014b; Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017), language modelling (Mnih and Teh, 2012; Al-Rfou et al., 2019), and parsing (Vinyals et al., 2015). Based on these successes, I present a neural network-based approach for morphological inflection.

The following chapter covers the background for understanding both the tasks which are the focus of this thesis as well as the machine learning models used to approach them. We first discuss the importance of handling morphology (2.1.1), introduce the morphological inflection and morphological reinflection tasks (2.1.2), and present main channels through which research in computational morphology is promoted (2.1.3). Then, we lay out the foundation for the models used in this thesis, and present an overview of several neural network architectures such as multi-layer perceptrons (2.2.1), recurrent neural networks (RNNs) (2.2.2) and recurrent neural sequence-to-sequence architectures (2.2.3). Finally, we present the transformer (2.2.5) and pointer-generator (2.2.6) network architectures, which are used in this thesis.

All the code used for our experiments can be found at: `https://github.com/AssafSinger94/sigmorphon-2020-inflection`

Chapter 2

Background

2.1 Morphology in NLP

2.1.1 The Importance of Handling Morphology

Different languages portray different levels of inflection. While some languages like English are morphologically impoverished with regards to inflection, others have many inflections per base form or lemma. For example, a Polish verb has nearly 100 inflected forms (Janecki, 2000) and an Archi verb has thousands (Kibrik, 1998). Those transformations of forms in morphologically rich languages yield a much larger vocabulary than in other languages. In such morphologically complex languages, approriate handling of morphology can reduce data sparsity. For example, statistical machine translation suffers from data sparsity when translating morphologically-rich languages, since every surface form is considered an independent entity. Translating into lemmas in the target language and then applying inflection generation as a post-processing step has been shown to alleviate the sparsity problem (Minkov et al., 2007; Clifton and Sarkar, 2011; Fraser et al.,

Lemma	Inflected form	Morphological tag
spend	spending	V;V.PTCP;PRS
spend	spends	V;3;SG;PRS
spend	spend	V;NFIN
spend	spent	V;PST
spend	spent	V;V.PTCP;PST

Figure 2.1: The paradigm of the English lemma *spend*, according to the notation defined in Section 2.1.2. The morphological tag is a bundle of morphosyntactic features.

2012). Modeling inflection generation has also been used to improve language modeling (Chahuneau et al., 2013b) and identification of multi-word expressions (Oflazer et al., 2004), among other applications.

2.1.2 Morphological Generation Tasks

In this thesis, we work on a group of closely related tasks concerning the generation of inflected forms. In order to formally describe our tasks, we define the following notation:

Let T be the set of all morphological tags being expressed in a language and l a lemma in the same language. The morphological paradigm π of lemma l is defined as follows:

$$\pi(l) = \{(f_k[l], t_k)\}_{k \in T(l)} \tag{2.1}$$

$f_k[l]$ denotes an inflected form corresponding to morphological tag t_k, and l and $f_k[l]$ are sequences of letters from the alphabet Σ of the language. Note that, even though we follow the convention to describe word forms as functions of the lemma, in the vast majority of the cases, each inflection is uniquely defined given any other inflected form of the same paradigm and the two respective tags.

Morphological inflection is the task of, given a lemma together with mor-

phosyntactic features defining the target form, generating the indicated inflected form. Formally, it is the task of predicting inflected form $f_k[l]$, given a lemma l together with a morphological tag t_k. An English example is:

$(spend, \text{V;V.PTCP;PRS}) \rightarrow spending$

Morphological reinflection is a generalized version of the morphological inflection task, which consists of producing an inflected form from any given source form – i.e., not necessarily the lemma –, and target tag. In our notation, it is the task of predicting a $f_j[l]$ from a paradigm, given a different form $f_i[l]$ in the paradigm or the lemma l, as well the target t_j and optionally the source tag t_i. This is a more complicated task, as the model needs to infer l of the source form in order to correctly inflect it to $f_j[l]$.

The final task is the one of **paradigm completion**. Given a partial paradigm $\pi(w)_p$ with $\pi(w)_p \subseteq \pi(w)$, the goal of paradigm completion is to produce all inflected forms $f_i[w]$ such that $(f_i[w], t_i) \in \pi(w)$, but $(f_i[w], t_i) \notin \pi(w)_p$. The corresponding tags are supposed to be known.

For example, consider the following partial paradigm, which is a subset of the English paradigm shown in Table 2.1:

$$\pi(spend)_p = \{(spends, \text{V;3;SG;PRS}), (spend, \text{V;NFIN})\}$$

We then expect a paradigm completion system to produce all unknown inflected forms, which correspond to the tags V;V.PTCP;PST, V;PST and V;V.PTCP;PRS.

2.1.3 The SIGMORPHON shared task

The ACL Special Interest Group on Computational Morphology and Phonology (SIGMORPHON) provides a forum for exchanging news of recent research developments and other matters of interest in the areas of computational morphology and phonology. As part of its activity, the group hosts yearly workshops with the purpose of bringing together researchers interested in applying computational techniques to problems in the field.

In 2016, the yearly SIGMORPHON shared task was launched, with the goal of promoting further research on computational morphology, focusing on varying settings for learning and analyzing inflectional patterns. In 2017 and 2018, SIGMORPHON partnered with the SIGNLL Conference on Computational Natural Language Learning (CONLL) to launch the CoNLL-SIGMORPHON shared task, before resuming to launch the task alone in 2019 and 2020. Many of the tasks include classic morphological generation in different setups such as morphological inflection and morphological reinflection (Cotterell et al., 2017a, 2018). These shared tasks have propelled research in computational morphology and many of the state-of-the-art models for morphological generation in recent years were presented by participants (Kann and Schütze, 2016b; Makarov et al., 2017; Makarov and Clematide, 2018). In an attempt to lead the development of morphological analysis in more general settings, some less traditional tasks are incorporated. Such as was the 2018 "Inflection in Context" cloze task (Cotterell et al., 2018) where participants were given a sentence with a number of missing word forms and corresponding lemmas with the goal of filling in the missing forms. Another example is Task 2 on unsupervised morphological paradigm completion of this year's shared task, where participants are given raw text in a given language and a

list of lemmas, with the objective the complete the entire paradigm of each lemma (Jin et al., 2020). The majority of the research presented in this thesis is based on the NYU–CUBoulder submissions created by Professor Katharina Kann and myself, to the SIGMORPHON 2020 Task 0 on typologically diverse morphological inflection and Task 2 on unsupervised morphological paradigm completion.

2.2 Neural Networks

An important family of machine learning models are neural networks, also known as deep learning. Neural networks have been inspired by biology, designed to mimic the behavior of our brain (Rosenblatt, 1957). A human brain contains an enormous amount of nerve cells, called neurons. Each of these cells is connected to many other similar cells, creating a very complex network of signal transmissions. Each cell collects inputs from all other neural cells it is connected to, and if the magnitude of its input reaches a certain threshold, it sends out a signal to all other cells it is connected to.

2.2.1 Perceptron and Multi-Layer Perceptron

One of the earliest attempts to mimic the behavior of a neuron was the perceptron (Rosenblatt, 1957). A perceptron processes several weighted inputs, sums them up, and, if this sum is above a threshold, produces an output signal. For a vector of inputs x, a perceptron can be formulated as:

$$Y = f(wx + b) \tag{2.2}$$

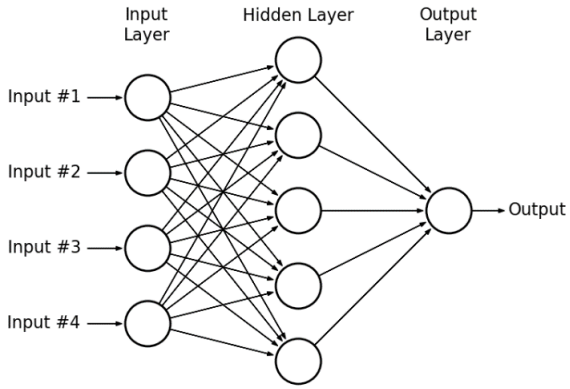


Figure 2.2: Multi-layer perceptron with one hidden layer. Figure by (Hassan et al., 2015)

where the nonlinear activation function $f()$ is given by a step function of the form

$$f(a) = \begin{cases} +1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases} \tag{2.3}$$

Y is the output signal, w is a weight vector, and b is a bias.

A neural network, also known as a multi-layer perceptron, is based on this concept. A multi-layer perceptron (MLP) consists of layers of nodes: an input layer, hidden layers and an output layer. Each node in a layer is computed as a weighted sum of the layer's inputs, passed through a nonlinear function. The nodes are then stacked together and passed on to the next layer, until reaching the output layer. More formally, given the vector x of stacked input nodes, each layer applies:

$$Y = g(Wx + b) \tag{2.4}$$

Where W_{i_j} connects input node i and output node j, and b is a bias vector. g is a nonlinear function, usually the sigmoid or ReLu functions in the hidden layer

and the softmax in the output layer. The network is also called a feed-forward neural network, due to this forward motion of the network. The model can get more complex as it grows deeper and contains more hidden layers. Neural network models are most commonly trained by minimizing a loss function, which accounts for errors, and updating the parameters via gradient descent.

2.2.2 Recurrent Neural Networks

While feed-forward neural networks are used for a variety of tasks, the fixed number of nodes in a network allows it to only handle inputs of constant size. Padding can be used to handle ranging-size data to a certain degree. Using padding, the model is defined to process a maximum size input and output (hopefully large enough to handle the majority of examples), where any data that is shorter than the maximum size is padded with zeros to fit the model's dimensions. This, however. is limiting as the network's input and output sizes cannot exceed the predetermined maximum size.

Recurrent neural networks (RNN) are a straightforward adaptation of the standard feed-forward neural network made to allow modeling of variable length sequences of input data. At each step, the model processes not only the current input, but also a representation of the data computed in previous model steps, implementing a sense of memory. This property makes RNNs highly suitable for dealing with sequential data such as natural language (Cho et al., 2014a; Sutskever et al., 2014; Al-Rfou et al., 2019) and music (Eck and Schmidhuber, 2002). The RNN architecture is as follows: given a sequence of input vectors $(x_1, \cdots, x_T)$, the network computes at each time step t a hidden state h_t, producing a sequence

$(h_1, \cdots, h_T)$, as follows:

$$h_t = g(W_h x_t + U_h h_{t-1} + b_h) \tag{2.5}$$

where matrices W_h, U_h and bias b_h are learnable. h_0 is either provided the user, set to zero or learned, and g is an element-wise nonlinear function. Once all inputs are processed, the model produces an output, either on top of each hidden state individually or on top of the last hidden state h_T. Gradients of the network are computed using backpropagation through time (BPTT). Recurrent neural networks are notoriously difficult to train because of the vanishing gradient problem. By application of the chain rule, derivatives are multiplied down the network (from the final to the initial layer) to compute the derivatives of the initial layers. However, when n hidden layers use an activation function like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers. Several model architectures were developed to approach this problem, such as the gated recurrent unit (GRU) network (Cho et al., 2014b), and the long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997) (which we will not describe here).

Introduced by Cho et al. (2014b), the **gated recurrent unit** has an update gate and a reset gate, which control the flow of information through the network. For input x_t and previous hidden state h_{t-1} at timestep t, the reset gate r_t and update gate z_t are computed as follows:

$$\begin{aligned} r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \end{aligned} \tag{2.6}$$

Here, σ is the sigmoid function, and weight matrices W_r, W_z, U_r, U_z and biases b_r, b_z can be learned. The reset weights are then used to compute the candidate hidden state $\tilde{h}_t$, and subsequently, h_t using the update weights.

$$\begin{aligned} \tilde{h}_t &= \tanh(W_{\tilde{h}} x_t + U_{\tilde{h}}(r_t h_{t-1}) + b_h) \\ h_t &= z_t h_{t-1} + (1 - z_t)\tilde{h}_t \end{aligned} \tag{2.7}$$

2.2.3 Encoder-Decoder Recurrent Neural Networks

For an input sequence $(x_1, \cdots, x_T)$, most classic RNN architectures either produce one output for the entire sequence, or compute an output for each input x_t. While beneficial for many tasks, such as classification or part of speech tagging, this forcibly binds the length of the output to that of the input. However, in many sequence-to-sequence tasks, such as machine translation, the sequence length of the input differs from the output. For example, the term *Hello world* in English is made of two words, differing in length from its three-words French translation *Bonjour le monde*. Cho et al. (2014b) approach this restriction developing a model which decouples the output sequence length from the input, allowing flexibility in both sequences. The model consists of two main components: an encoder and a decoder. The encoder's role is to process the input and abstract the relevant information, which the decoder then uses to generate the output. For input sequence $(x_1, \cdots, x_{T_x})$, the encoder, computes the hidden states $(h_1, \cdots, h_{T_x})$ as follows:

$$h_t = f(x_t, h_{t-1}) \tag{2.8}$$

Where f is modeled by a GRU, and eventually outputs a context vector c, holding meaningful information about the input.

$$c = m\left(h_1, \cdots, h_{T_x}\right) \tag{2.9}$$

M can be any function that utilizes the hidden states, where in the model by Cho et al. (2014b) simply outputs the last hidden state h_{T_x}. The decoder, implemented as a second RNN, then uses the context vector to generate the output in a sequential manner. At each step t, the model predicts output y_t based on all previous outputs, as well as the context vector.

$$p(y) = \prod_{t=1}^{T_y} p(y_t | y_1, \cdots, y_{t-1}, c) \tag{2.10}$$

Where p is a conditional probability modeled by an RNN.

$$p(y_t | y_1, \cdots, y_{t-1}, c) = g(y_{t-1}, s_t, c) \tag{2.11}$$

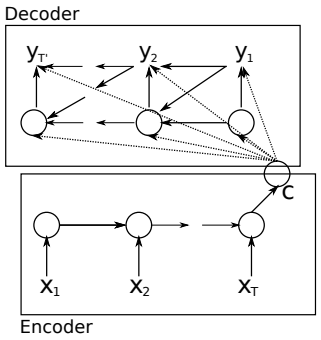
Again, g is modeled by a GRU and s_t is the hidden state of the decoder RNN.

2.2.4 Attention

While the input and output sequences may vary in length, the context vector remains constant in size. This creates an information bottleneck, and Cho et al. (2014a) show that it decreases the performance of models as the length of the input increases. To alleviate this issue, Bahdanau et al. (2015) propose an encoder-decoder model based on a novel concept known as attention.

This model is largely identical to the previous encoder-decoder model. However,

Encoder-decoder RNN



Attention-based encoder-decoder RNN

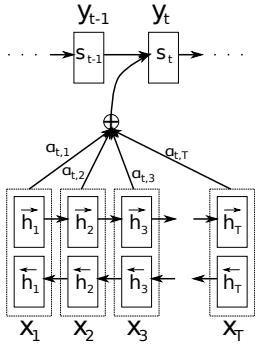


Figure 2.3: Left: the RNN encoder–decoder. Figure by (Cho et al., 2014b) Right: the attention-based encoder-decoder model architecture. Figure by (Bahdanau et al., 2015)

in order to avoid the constraint of the single fixed-length context vector c, the output distribution is conditioned on a distinct context vector c_i for each target word y_i as follows.

$$p(y_i|y_1, \cdots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \tag{2.12}$$

c_i is computed as a weighted sum of the hidden states produced by the encoder:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.13}$$

Where the attention weight α_{ij} for each annotation h_j is computed by:

$$\begin{aligned} e_{ij} &= a(s_{i-1}, h_j) \\ \alpha_{ij} &= \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \end{aligned} \tag{2.14}$$

where a is an alignment model which scores how well the input at position j and the output at position i match, implemented using a feedforward neural network. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i) and

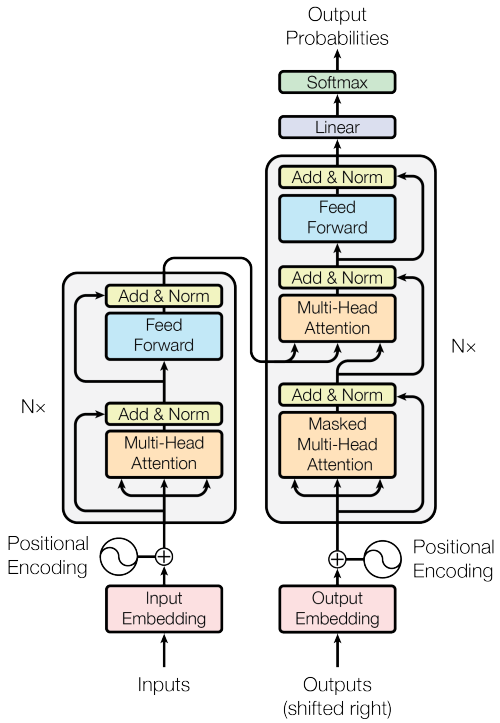


Figure 2.4: The transformer: model architecture. Figure by (See et al., 2017)

hidden state h_j. We parameterize the alignment model as a feed-forward neural network which is jointly trained with all the other components of the proposed system, where the attention weights determine how strongly the current output y_t is affected by the input x_j.

To model each h_i to obtain information about the surrounding positions, the encoder RNN is implemented as a bi-directional RNN. The bi-directional RNN is composed of a forward and a backward RNN, producing $(h_1^{\Rightarrow}, \cdots, h_T^{\Rightarrow})$, and $(h_1^{\Leftarrow}, \cdots, h_T^{\Leftarrow})$, which are concatenated to produce hidden states $(h_1, \cdots, h_T)$.

2.2.5 Transformer

One major setback is that RNNs take a long time to train. At each time step t, an RNN must know the previous hidden state h_{t-1} when producing h_t.

This sequential nature imposes a computational bottleneck, which precludes parallelization within training examples. This issue becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recently, a new model architecture was introduced for sequence-to-sequence tasks, called the transformer (Vaswani et al., 2017). It is based solely on attention mechanisms, dispensing with recurrence entirely, offering more parallelization and requiring significantly less time to train. Transformers have produced state-of-the-art results on various tasks, such as machine translation (Vaswani et al., 2017) language modeling (Al-Rfou et al., 2019), question answering (Devlin et al., 2019) and language inference (Devlin et al., 2019).

The transformer consists of an encoder and a decoder, each composed of a stack of layers. Given an input sequence $(x_1, \cdots, x_T)$, the encoder processes the entire sequence at once to produce hidden states $(h_1, \cdots, h_T)$. At generation step t, the decoder reads the previously generated sequence $(y_1, \cdots, y_{t-1})$ to produce states $(s_1, \cdots, s_{t-1})$. During training, the entire target sequence $(y_1, \cdots, y_{Ty})$ is input to the decoder at once, along with a sequential mask used to prevent positions from attending to subsequent positions. In order to differentiate between the different positions, a sinusoidal positional encoding is added to the input and output sequences, which the model then learns to recognize.

2.2.5.1 Encoder and Decoder Stacks

The architecture of the encoder and decoder stacks is shown in Figure 2.4. The encoder and decoder are both composed of a stack of N subsequent layers. The encoder layers consist of a self-attention layer, followed by a fully connected layer, and the decoder layers contain an additional inter-attention layer between

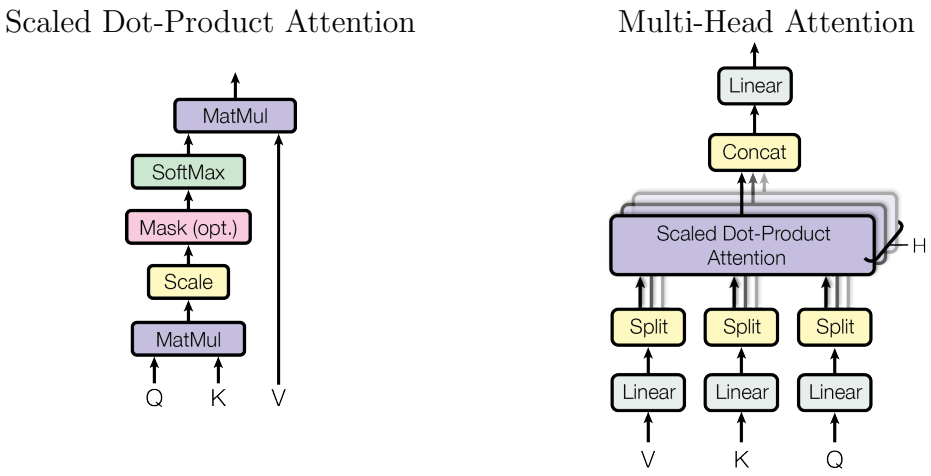


Figure 2.5: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Figure by (Vaswani et al., 2017)

the two. Using self-attention helps the model explore the alignment between the different positions within the same sequence as it encodes a specific word, where the inter-attention acts as an alignment model which measures how well the different positions of the input sequence and the output sequence align. Another way in which the decoder differs from the encoder is that the self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Residual connections are applied after every sub-layer, along with layer normalization. The sequence length and embedding dimensions do not change between each sub-layer and between layers and stay the same through the entire network.

2.2.5.2 Scaled dot-product Attention

Attention can be seen as aligning a set of queries with a set of key-value pairs. For a given query q, the output is a weighted average of the values, based on the alignment between the query and the corresponding keys. In scaled-dot-product attention, for queries and keys of dimension d_k, and values of dimension d_v, we compute the dot products of the query with all keys, and apply a softmax function to obtain the weights on the values. To allow for parallel attention computation, the queries, keys and values are stacked together, producing all outputs at once as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.15}$$

In order to prevent the dot products from growing large in magnitude, the product is then scaled down by $\sqrt{d_k}$. For the inter-attention which seeks to align the input and output sequences, the inputs to the decoder layer are used as queries, and the encoder hidden states are used as both the keys and values. In self-attention, where we align the different positions of the same sequence, the inputs to the layer are used as the queries, keys and values.

2.2.5.3 Multi-Headed Attention

In order to allow the model to learn different representations of the inputs, the embeddings are projected to different sub-spaces, as follows. Each attention layer is divided to h separate heads, where at each head, the inputs of the layer are projected to h different sub-spaces of dimension d_k. The model then applies the attention between the projected embeddings, eventually concatenating all heads

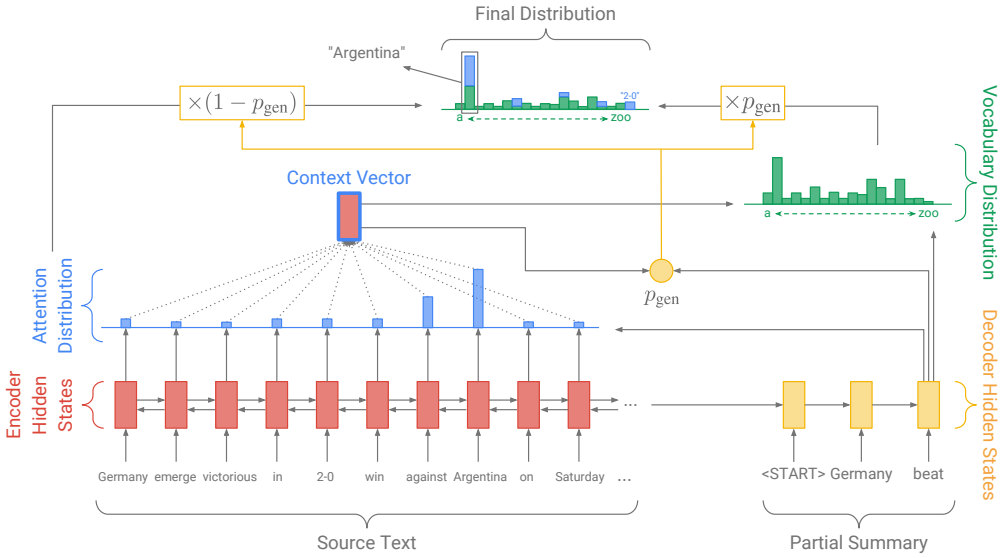


Figure 2.6: Pointer-generator model. For each decoder timestep a generation probability $p_{\text{gen}} \in [0, 1]$ is calculated, which weights the probability of *generating* words from the vocabulary, versus *copying* words from the source text. The vocabulary distribution and the attention distribution are weighted and summed to obtain the final distribution, from which we make our prediction. Figure by See et al. (2017)

together. Formally, the multi-head attention layer is defined as follows:

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, \cdots, \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d \times d_k}$, $W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d}$, and d is the input dimension.

2.2.6 Pointer-Generator Network

Copying of pieces from the input in order to generate an output has been proved as beneficial to some natural language tasks. A popular example is extractive summarization, where models extract words from an article to form its summary. This

use of copying was also implemented by the pointer network (Vinyals et al., 2015), which constructs an output by pointing to different positions of the input. This concept was then extended by See et al. (2017), who introduce a pointer-generator network for abstractive summarization. The pointer-generator network can copy words from the source text via pointing, which facilitates accurate reproduction of information, while retaining the ability to produce novel words. This model has sometimes been enhanced by a coverage mechanism, which helps keep track of previously generated outputs, in order to reduce repetition during generation.

The pointer-generator transformer uses a combination of a "generation" distribution and a "copy" distribution, which allows for both generating characters from a fixed vocabulary, as well as copying from the source text via pointing. This is managed by the generation probability $p_{\text{gen}} \in [0, 1]$ which acts as a soft switch between the two actions.

The model is constructed as an extension over the attention-based encoder-decoder network by Bahdanau et al. (2015), with the encoder and decoder implemented with an LSTM instead of a GRU. Given inputs $(w_1, \cdots, w_T)$, the attention-based encoder-decoder model produces the "generational" probability at timestep t using a distribution over the output vocabulary:

$$P_{\text{vocab}} = \text{softmax}(V'(V[s_t, h_t^*] + b) + b') \tag{2.16}$$

Then, p_{gen} is calculated from the context vector h_t, the decoder state s_t and the decoder input x_t:

$$p_{\text{gen}} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{gen}) \tag{2.17}$$

where vectors w_{h^*}, w_s, w_x and scalar b_{ptr} are learnable parameters and σ is the sigmoid function. For each document let the extended vocabulary denote the union of the vocabulary, and all words appearing in the source document. The probability distribution over the extended vocabulary is computed by:

$$P(w) = p_{\text{gen}} P_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} a_i^t \tag{2.18}$$

Note that if l is an out-of-vocabulary (OOV) word, then $P_{\text{vocab}}(w)$ is zero, and if l does not appear in the source document, then $\sum_{i:w_i=w} a_i^t$ is also zero. The ability to produce OOV words is one of the primary advantages of pointer-generator models; by contrast models such as our baseline are restricted to their pre-set vocabulary.

Chapter 3

Research

3.1 Approach Details

In this thesis, we suggest new approaches to problems in inflectional morphology, more specifically morphological inflection and paradigm completion. We model morphological inflection as a sequence-to-sequence problem, where the input is the sequence of the lemma's characters with morphological tags, and the output is the sequence of the inflected form's characters. First, we apply a transformer model to the task. Then, we present the pointer-generator transformer model, based on the vanilla transformer model (Vaswani et al., 2017) and the pointer-generator model See et al. (2017). After adding a copy mechanism to the transformer, it produces a final probability distribution as a combination of generating elements from its output vocabulary and copying elements – characters in our case – from the input. In order to examine our proposed model, we participate in Task 0 of the SIGMORPHON 2020 shared task on typologically diverse morphological inflection and Task 2 on unsupervised morphological paradigm completion.

Task 0 consists of generating morphological inflections from a lemma and a set of morphosyntactic features describing the target form. For our submissions, we further increase the size of all training sets by performing multi-task training on morphological inflection and morphological reinflection, i.e., the task of generating inflected forms from forms *different from the lemma*. . For languages with small training sets, we also perform hallucination pretraining (Anastasopoulos and Neubig, 2019), where we generate pseudo training instances for the task, based on suffixation and prefixation rules collected from the original dataset.

For **Task 2**, participants are given raw text and a source file with lemmas. The objective is to generate the complete paradigms for all lemmas. Our systems for this task consist of a combination of the official baseline system (Jin et al., 2020) and our systems for Task 0. The baseline system finds inflected forms in the text, decides on the number of inflected forms per lemma, and produces pseudo training files for morphological inflection. Our inflection model then learns from these and, subsequently, generates all missing forms.

As most inflected forms derive their characters from the source lemma, the use of a mechanism for copying characters directly from the lemma has proven to be effective for morphological inflection generation, especially in the low resource setting (Aharoni and Goldberg, 2017; Makarov et al., 2017). As all Task 0 datasets are fairly large, we further design a low-resource experiment to investigate the effectiveness of our model.

3.2 Related Work

3.2.1 Morphological Generation

In recent years, the SIGMORPHON and CoNLL–SIGMORPHON shared tasks have promoted research on computational morphology, with a strong focus on morphological inflection and reinflection. Research related to those shared tasks includes Kann and Schütze (2016a), who used an LSTM (Hochreiter and Schmidhuber, 1997) sequence-to-sequence model with soft attention (Bahdanau et al., 2015) and achieved the best result in the SIGMORPHON 2016 shared task (Kann and Schütze, 2016b; Cotterell et al., 2016a). Due to the often monotonic alignment between input and output in the tasks of morphological generation, Aharoni and Goldberg (2017) proposed a model with hard monotonic attention. Unlike sequence-to-sequence models based on soft attention (Bahdanau et al., 2015), this model attends to only a single input state at a time and either adds a symbol to the output sequence or moves the position of the attention pointer to the next hidden state of the encoder. Based on this, Makarov et al. (2017) implemented a neural state-transition system which also used hard monotonic attention to transduce the lemma into the inflected form by a sequence of explicit edit operations, and achieved the best results for Task 1 of the SIGMORPHON 2017 shared task. In 2018, the best results were achieved by a revised version of the neural transducer, trained with imitation learning (Makarov and Clematide, 2018). That model learned an alignment instead of maximizing the likelihood of gold action sequences given by a separate aligner.

For paradigm completion, extended work has been done for Task 2 of the 2017 shared task. Cotterell et al. (2017b) considered a multi-source setting for paradigm

completion. They modeled paradigms using graphical models with neural parameterizations, defined over multiple string-valued random variables.

Earlier influential work on paradigm completion or inflection— both neural and non-neural—which is not mentioned above, included but was by no means limited to (Dreyer et al., 2008; Faruqui et al., 2016; Hulden et al., 2014)

3.2.2 Sequence-to-Sequence Models in NLP

In the last few years, sequence-to-sequence models have been found useful to several natural language processing (NLP) tasks, limited not only to machine translation (Cho et al., 2014a; Sutskever et al., 2014; Bahdanau et al., 2015) but also to parsing (Vinyals et al., 2015), speech recognition (Graves and Schmidhuber, 2005) and many more tasks.

3.2.3 Transformers

Transformers have produced state-of-the-art results on various tasks such as machine translation (Vaswani et al., 2017) and language modeling (Al-Rfou et al., 2019). Several revisions have been done to the model, most notably is BERT Devlin et al. (2019), which stands for Bidirectional Encoder Representations from Transformers. BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. BERT achieved state-of-the-art results on several tasks such as question answering and language understanding (Devlin et al., 2019). There has been very little work on transformers for morphological inflection, with, to the best of our knowledge, Erdmann et al. (2020) being the only published paper. However, the widespread success of transformers in NLP leads us to believe that a transformer

model could perform well on morphological inflection.

3.2.4 Pointer-Generators

In addition to the transformer, the architecture of our model is also inspired by See et al. (2017), who used a pointer-generator network for abstractive summarization. Their model could choose between generating a new element and copying an element from the input directly to the output. This copying of words from the source text via pointing (Vinyals et al., 2015), improved the handling of out-of-vocabulary words. Copy mechanisms have also been used for other tasks, including morphological inflection (Sharma et al., 2018). *Transformers* with copy mechanisms have been used for word-level tasks (Zhao et al., 2019), but, as far as we know, never before on the character level.

3.3 SIGMORPHON 2020 Shared Task

The SIGMORPHON 2020 Shared Task is composed of three tasks: Task 0 on typologically diverse morphological inflection, Task 1 on multilingual grapheme-to-phoneme conversion, and Task 2 on unsupervised morphological paradigm completion. We submit systems to Tasks 0 and 2.

3.3.1 Task 0: Typologically Diverse Morphological Inflection

SIGMORPHON 2020 Task 0 focuses on morphological inflection in a set of typologically diverse languages. Different languages inflect differently, so it is not trivially clear that systems that work on some languages also perform well on

others. For Task 0, systems need to generalize well to a large group of languages, including languages unseen during model development.

The task features 90 languages in total. 45 of them are development languages, coming from five families: Austronesian, Niger–Congo, Uralic, Oto-Manguean, and Indo-European. The remaining 45 are surprise languages, and many of those are from language families different from the development languages. Some languages have very small training sets, which makes them hard to model. For those cases, the organizers recommend a family-based multilingual approach to exploit similarities between related languages. While this might be effective, we believe that using multitask training in combination with hallucination pretraining can give the model enough information to learn the task well, while staying true to the specific structure of each individual language.

3.3.2 Task 2: Unsupervised Morphological Paradigm Completion

Task 2 is a novel task, designed to encourage work on unsupervised methods for computational morphology. As morphological annotations are limited for many of the world's languages, the study of morphological generation in the low-resource setting is of great interest (Cotterell et al., 2018). However, a different way to tackle the problem is by creating systems that are able to use data without annotations.

For Task 2, a tokenized Bible in each language is given to the participants, along with a list of lemmas. Participants should then produce complete paradigms for each lemma. As slots in the paradigm are not labeled with gold data paradigm slot descriptions, an evaluation metric called best-match accuracy was designed for this task. First, this metric matches predicted paradigm slots with gold slots in the

way which leads to the highest overall accuracy. It then evaluates the correctness of individual inflected forms.

3.4 Methods

In this section, we introduce our models for Tasks 0 and 2 and describe all approaches we use, such as multitask training, hallucination pretraining and ensembling.

3.4.1 Transformer

With inputs $(x_1, \cdots, x_T)$ being a lemma's characters followed by tags representing the morphosyntactic features of the target form, the encoder processes the input sequence and outputs hidden states $(h_1, \cdots, h_T)$. At generation step t, the decoder reads the previously generated sequence $(y_1, \cdots, y_{t-1})$ to produce states $(s_1, \cdots, s_{t-1})$. The last decoder state s_{t-1} is then passed through a linear layer followed by a softmax, to generate a probability distribution over the output vocabulary:

$$P_{\text{vocab}} = \text{softmax}(V s_{t-1} + b) \tag{3.1}$$

During training, the entire target sequence $\left(y_1, \cdots, y_{T_y}\right)$ is input to the decoder at once, along with a sequential mask to prevent positions from attending to subsequent positions.

3.4.2 Pointer-Generator Transformer

The pointer-generator transformer allows for both generating characters from a fixed vocabulary, as well as copying from the source sequence via pointing (Vinyals et al., 2015). This is managed by p_{gen} – the probability of generating as opposed to copying – which acts as a soft switch between the two actions. p_{gen} is computed by passing a concatenation of the decoder state s_t, the previously generated output y_{t-1}, and a context vector c_t through a linear layer, followed by the sigmoid function.

$$p_{\text{gen}} = \sigma(w[s_t; c_t; y_{t-1}] + b) \tag{3.2}$$

The context vector is computed as the weighted sum of the encoder hidden states

$$c_t = \sum_{i=1}^{T} a_i^t h_i \tag{3.3}$$

with attention weights $(a_1^t, \cdots, a_T^t)$. For each inflection example, let the extended vocabulary denote the union of the output vocabulary, and all characters appearing in the source lemma. We then use p_{gen}, P_{vocab} produced by the transformer, and the attention weights of the last decoder layer $(a_1^t, \cdots, a_T^t)$ to compute a distribution over the extended vocabulary:

$$P(c) = p_{\text{gen}} P_{\text{vocab}}(c) + (1 - p_{\text{gen}}) P_{\text{copy}}(c) \tag{3.4}$$

with

$$P_{\text{copy}}(c) = \sum_{i:x_i=c} a_i^t \tag{3.5}$$

raw	grip	grips	V;SG;3;PRS
	grip	gripped	V;PST
	grips	grip	V;LEMMA
generated	grips	gripped	V;PST
	gripped	grip	V;LEMMA

Figure 3.1: An English multitask training example.

The copy distribution $P_{\text{copy}}(c)$ for each character c is the sum of attention weights over all source positions where $x_i = c$. Note that if c is an out-of-vocabulary (OOV) character, then $P_{\text{vocab}}(c)$ is zero; similarly, if c does not appear in the source lemma, then $\sum_{i:x_i=c} a_i^t$ is zero. The ability to produce OOV characters is one of the primary advantages of pointer-generator models; by contrast models such as our vanilla transformer are restricted to their pre-set vocabulary.

3.4.3 Multitask Training

Some languages in Task 0 have small training sets, which makes them hard to model. In order to handle that, we perform multitask training, and, thereby, increase the amount of examples available for training.

Morphological reinflection. Morphological reinflection is a generalized version of the morphological inflection task, which consists of producing an inflected form for any given source form – i.e., not necessarily the lemma –, and target tag. For example: $(hugging; V; PST) \rightarrow hugged$.

This is a more complex task, since a model needs to infer the underlying lemma of the source form in order to inflect it correctly to the desired form. Many morphological inflection datasets contain lemmas that are converted to several inflected forms. Treating separate instances for the same source lemma as independent is

missing an opportunity to utilize the connection between the different inflected forms. We approach this by converting our morphological inflection training set into one for morphological reinflection as described in the following.

From inflection to reinflection. Inflected forms of the same lemma are grouped together to sets of one or more (inflected form, morphological features) pairs. Then, for each set, we create new training instances by inflecting all forms to one another, as shown in Figure 3.1. We also let the model inflect forms back to the lemma by adding the lemma as one of the inflected forms, marked with the synthetically generated LEMMA tag. The new training set fully utilizes the connections between different forms in the paradigm, and, in that way, provides more training instances to our model.

3.4.4 Hallucination Pretraining

Another effective tool to improve training in the low-resource setting is data hallucination (Anastasopoulos and Neubig, 2019). Using hallucination, new pseudo-instances are generated for training, based on suffixation and prefixation rules collected from the original dataset. For languages with less than 1000 training instances, we pretrain our models on a hallucinated training set consisting of 10,000 instances, before training on the multitask training set.

3.4.5 Submissions and Ensembling Strategies

We submit 4 different systems for Task 0. NYU-CUBoulder-2 consists of one pointer-generator transformer model, and, for NYU-CUBoulder-4, we train one

Hyperparameter	Value
Embedding dimension	256
Encoder layers	4
Decoder layers	4
Encoder hidden dimension	1024
Decoder hidden dimension	1024
Attention heads	4

Table 3.1: The hyperparameters used in our models.

vanilla transformer. Those two are our simplest systems and can be seen as baselines for our other submissions.

Because of the effects of random initialization in non-convex objective functions, we further use ensembling in combination with both architectures: NYU-CUBoulder-1 is an ensemble of three pointer-generator transformers, and NYU-CUBoulder-3 is an ensemble of five pointer-generator transformers. The final decision is made by majority voting. In case of a tie, the answer is chosen randomly among the most frequent predictions. Models participating in the ensembles are from different epochs during the same training run.

As previously stated, all systems are trained on the augmented multitask training sets, and systems trained on languages with less than 1000 training instances were pretrained on the hallucinated datasets.

3.4.6 Task 2: Model description

Our systems for Task 2 consist of a combination of the official baseline system (Jin et al., 2020) and our inflection systems for Task 0. The system is given raw text and a source file with lemmas, and generates the complete paradigm of each lemma. The baseline system finds inflected forms in the text, decides on the number of inflected forms per lemma, and produces pseudo training files for

morphological inflection. Any inflections that the system has not found in the raw text are given as test instances. Our inflection model then learns from the files and, subsequently, generates all missing forms. We use the pointer-generator and vanilla transformers as our inflection models.

For Task 2, we use ensembling for all submissions. NYU-CUBoulder-1 is an ensemble of six pointer-generator transformers, NYU-CUBoulder-2 is an ensemble of six vanilla transformers, and NYU-CUBoulder-3 is an ensemble of all twelve models. For all models in both tasks, we use the hyperparameters described in Table 3.1.

3.5 Experiments

3.5.1 Task 0

Data. The dataset for Task 0 covers 90 languages in total: 45 development languages and 45 surprise languages. For details on the official dataset please refer to Vylomova et al. (2020).

Baselines. This year, several baselines are provided for the task. The first system has also been used as a baseline in previous shared tasks on morphological reinflection (Cotterell et al., 2017a, 2018). It is a non-neural system which first scans the dataset to extract suffix- or prefix-based lemma-to-form transformations. Then, based on the morphological tag at inference time, it applies the most frequent suitable transformation to an input lemma to yield the output form (Cotterell et al., 2017a). The other two baselines are neural models. One is a transformer (Vaswani et al., 2017), and the second one is a hard-attention model

System:	Sub-1	Sub-2	Sub-3	Sub-4	Base
Development Set					
Low	**88.71**	88.02	84.90	84.07	-
Other	90.46	90.63	90.20	**90.94**	-
All	**90.06**	90.02	88.96	89.34	-
Test Set					
Low	84.8	84.8	85.5	83.9	**89.77**
Other	89.7	89.8	89.8	90.2	**92.43**
All	88.6	88.7	88.8	88.8	**91.81**

Table 3.2: Macro-averaged results over all languages on the official development and test sets for Task 0. Low=languages with less than 1000 train instances, Other=all other languages, All=all languages.

(Wu and Cotterell, 2019), which enforces strict monotonicity and learns a latent alignment while learning to transduce. To account for the low-resource settings for some languages, the organizers also employ two additional methods: constructing a multilingual model trained for all languages belonging to each language family **?**, and data augmentation using hallucination (Anastasopoulos and Neubig, 2019). Four model types are trained for each neural architecture: a plain model, a family-multilingual model, a data augmented model, and an augmented family-multilingual model. Overall, there are nine baseline systems for each language. We compare our models to an oracle baseline by choosing the best score over all baseline systems for each language.

Results. Our results for Task 0 are displayed in Table 3.2. All four systems produce relatively similar results. NYU-CUBoulder-3, our five-model ensemble, performs best overall with 88.8% accuracy on average. We further look at the results for low-resource ($<$ 1000 training examples) and high-resource ($>=$ 1000 training examples) languages separately. This way, we are able to see the advantage of the pointer-generator transformer in the low-resource setting, where all

pointer-generator systems achieve an at least 0.9% higher accuracy than the vanilla transformer model. However, in the setting where training data is abundant, the effect of the copy mechanism vanishes, as NYU-CUBoulder-4 – our only vanilla transformer – achieved the best results for our high-resource languages.

3.5.2 Task 2

Data. For Task 2, a tokenized Bible in each language is given to the participants, along with a list of lemmas. Participants are required to construct the paradigms for all given lemmas.

The languages for Task 2 are again divided into development and test languages. Development languages are available for model development and hyperparameter tuning, but are not used during the final evaluation. The test languages are used for evaluation only, and do not have development sets. The development languages are: Maltese, Persian, Portuguese, Russian, Swedish. The test languages are: Basque, Bulgarian, English, Finnish, German, Kannada, Navajo, Spanish and Turkish.

Baselines. The baseline system for the task is composed of four components, eventually producing morphological paradigms. The first three modules perform edit tree (Chrupala, 2020) retrieval, additional lemma retrieval from the corpus, and paradigm size discovery, using distributional information. After the first three steps, pseudo training and test files for morphological inflection are produced. Finally, the non-neural Task 0 baseline system Cotterell et al. (2017a) or the neural transducer by Makarov and Clematide (2018) are used to create missing inflected forms.

System	Baseline 1		Baseline 2		Sub-1		Sub-2		Sub-3	
	Test Set									
	slots	macro	slots	macro	slots	macro	slots	macro	slots	macro
Basque	30	0.0006	27	0.0006	30	0.0005	30	0.0005	30	**0.0007**
Bulgarian	35	0.283	34	**0.3169**	35	0.2769	35	0.2894	35	0.2789
English	4	0.656	4	**0.662**	4	0.502	4	0.528	4	0.512
Finnish	21	0.0533	21	**0.055**	21	0.0536	21	0.0547	21	0.0535
German	9	0.2835	9	**0.29**	9	0.273	9	0.2735	9	0.2735
Kannada	172	0.1549	172	**0.1512**	172	0.111	172	0.1116	172	0.111
Navajo	3	0.0323	3	**0.0327**	3	0.004	3	0.0043	3	0.0043
Spanish	29	0.2296	29	**0.2367**	29	0.2039	29	0.2056	29	0.203
Turkish	104	0.1421	104	**0.1553**	104	0.1488	104	0.1539	104	0.1513
All		0.2039		**0.2112**		0.1749		0.1802		0.1765

Table 3.3: Results for all test languages on the official test sets for Task 2.

Results. Systems for Task 2 are evaluated using macro-averaged best-match accuracy (Jin et al., 2020). Results are shown in in Table 3.3. All three systems produce relatively similar results. NYU-CUBoulder-2, our vanilla transformer ensemble, performed slightly better overall with an average best-match accuracy of 18.02%. Since our system is close to the baseline models, it performs similarly, achieving slightly worse results. For Basque, our all-round ensemble NYU-CUBoulder-2 outperformed both baselines with a best-match accuracy of 00.07%, achieving the highest result in the shared task.

3.5.3 Low-resource Setting

As most inflected forms derive their characters from the source lemma, the use of a mechanism for copying characters directly from the lemma has proven to be effective for morphological inflection generation, especially in the low-resource setting (Aharoni and Goldberg, 2017; Makarov et al., 2017). As all Task 0 datasets are fairly large, we further design a low-resource experiment to investigate the effectiveness of our model.

Data. We simulate a low-resource setting by sampling 100 instances from all languages that we already consider low-resource, i.e., all languages with less than 1000 training instances. We then keep their development and test sets unchanged. Overall, we perform this experiment on 21 languages.

Experimental setup. We train a pointer-generator transformer and a vanilla transformer on the modified datasets to examine the effects of the copy mechanism. We keep the hyperparameters unchanged, i.e., they are as mentioned in Table 3.1. We use a majority-vote ensemble consisting of 5 individual models for each architecture.

Baseline. We additionally train the neural transducer by Makarov and Clematide (2018), which has achieved the best results for the 2018 shared task in the low-resource setting (Cotterell et al., 2018). The neural transducer uses hard monotonic attention (Aharoni and Goldberg, 2017) and transduces the lemma into the inflected form by a sequence of explicit edit operations. It is trained with an imitation learning method (Makarov and Clematide, 2018). We use this model as a reference for the state of the art in the low-resource setting.

System	Trm	Trm-PG	Baseline
All	63.06	67.61	**70.06**

Table 3.4: Results on the official development data for our low-resource experiment. Trm=Vanilla transformer, Trm-PG=Pointer-generator transformer, Baseline=neural transducer by Makarov and Clematide (2018).

Results. As seen in Table 3.4, for the low-resource dataset, the pointer-generator transformer clearly outperforms the vanilla transformer by an average accuracy of 4.46%. For some languages, such as Chichicapan Zapotec, the difference is up to

Model:	1	2	3	4	5
Copy	✓	✓			✓
Multitask Train	✓		✓		✓
Hallucination	✓	✓	✓	✓	

Table 3.5: System components for the ablation study. Each model is a transformer which contains a combination of the following components: copy mechanism, multitask training and hallucination pretraining.

14%. While the neural transducer achieves a higher accuracy, our model performs only 2.45% worse than this state-of-the-art model.[1] We are also able to observe the use of the copy mechanism for copying of OOV characters in the test sets of some languages.

3.6 Ablation Studies

Our systems use three components on top of the vanilla transformer: a copy mechanism, multitask training and hallucination pretraining. We further perform an ablation study to measure the contribution of each component to the overall system performance. For this, we additionally train five different systems with different combinations of components. A description of which component is used in which system for this ablation study is shown in Table 3.5.

3.6.1 Results

Copy mechanism. Comparing models 2 and 4, which are both trained on the original dataset, pretrained with hallucination and differ only by the use of the copy mechanism, we are able to see that adding this component slightly improves performance by $0.06 - 0.16\%$. When comparing models 1 and 3, the copy mecha-

[1] We could probably obtain better results with appropriate hyperparameter tuning.

nism decreases performance slightly by 0.3% for the high-resource languages and 0.11% overall, but increases performance for low-resource languages by 0.68%.

Multitask training. Unlike the copy mechanism, multitask training actually consistently decreases the performance of the models. Looking at models 1 and 2, training the pointer-generator transformer on the multitask dataset decreases accuracy by $1.8 - 2.03\%$ for all three language groups. The same happens for the vanilla transformer with an accuracy decrease of $1.67 - 2.32\%$. A possible explanation are the relatively large training sets provided for the shared task, as this method is more suitable for the low-resource setting.

Hallucination pretraining. In order to examine the effect of hallucination pretraining on our submitted models, we now compare the pointer-generator transformers trained on the multitask data with and without hallucination pretraining (models 1 and 5). Hallucination pretraining shows to be helpful: it increases the accuracy on low-resource languages by 1.85%. The performance on the high-resource languages is necessarily the same, as only models for low-resource languages are actually pretrained.

Model:	1	2	3	4	5
Development Set					
Low	88.20	**90.00**	87.52	89.84	86.35
Other	90.63	**92.66**	90.93	92.60	90.63
All	90.02	**92.04**	90.13	91.96	89.63

Table 3.6: Ablation study; development set results, averaged over all languages. Low=languages with less than 1000 train instances, Other=all other languages, All=all languages.

Chapter 4

Conclusions

In this thesis, we suggested new approaches to problems in inflectional morphology. Specifically, we focused on the following questions: 1. How can we create morphological inflection systems that generalize well to a large group of languages? 2. How can we assist morphological inflection systems to perform in settings where training data is limited? 3. How can we utilize data that is not annotated to infer complete paradigms?

In Chapter 2 we first provided some background on the importance of handling inflection in natural language, defined the different morphological inflection generation tasks, and presented the SIGMORPHON shared task, founded for the purpose of promoting research on computational approaches to inflectional morphology.

We then introduced the feed-forward neural network model, discussed its limitations, and presented the recurrent neural network designed to handle sequential data. We then discussed several recurrent neural sequence-to-sequence architectures and the concept of attention (Bahdanau et al., 2015) as a tool to enable systems to attend to different positions in the source data when making predictions.

Finally, we present the transformer (Vaswani et al., 2017) and pointer-generator (See et al., 2017) network architectures, which are used in this thesis.

We presented the NYU-CUBoulder submissions for SIGMORPHON 2020 Task 0 of typologically diverse morphological inflection and Task 2 of unsupervised morphological paradigm completion (Vylomova et al., 2020).

We developed morphological inflection models, based on a transformer and a new model for the task, a pointer-generator transformer, which is a transformer-analogue of a pointer-generator model. For Task 0, we further added multitask training and hallucination pretraining, with the goal of improving performance in low-resource settings. For Task 2, we combined our inflection models with additional components from the provided baseline (Jin et al., 2020) to obtain a fully functional system for unsupervised morphological paradigm completion.

We performed an ablation study to examine the effects of all components of our inflection system. Finally, we designed a low-resource experiment to show that using the copy mechanism on top of the vanilla transformer is beneficial if training sets are small, and achieved results close to a state-of-the-art model for low-resource morphological inflection (Makarov and Clematide, 2018; Cotterell et al., 2018).

Overall, we hope that this thesis will make a contribution to research on morphology generation and analysis, and help to make neural network models applicable in low-resource settings.

Bibliography

R. Aharoni and Y. Goldberg. Morphological inflection generation with hard monotonic attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 2004–2015, 2017. doi: 10.18653/v1/P17-1183.

R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones. Character-level language modeling with deeper self-attention. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019.*, pages 3159–3166, 2019. doi: 10.1609/aaai.v33i01.33013159.

A. Anastasopoulos and G. Neubig. Pushing the limits of low-resource morphological inflection. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 984–996. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1091.

S. R. Anderson. *Morphology.* Encyclopedia of Cognitive Science. Macmillan Reference, Ltd., Yale University., 07 2016.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

V. Chahuneau, E. Schlinger, N. A. Smith, and C. Dyer. Translating into morphologically rich languages with synthetic phrases. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1677–1687. ACL, 2013a.

V. Chahuneau, N. A. Smith, and C. Dyer. Knowledge-rich morphological priors for bayesian language models. In L. Vanderwende, H. D. III, and K. Kirchhoff, editors, *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 1206–1215. The Association for Computational Linguistics, 2013b.

K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In D. Wu, M. Carpuat, X. Carreras, and E. M. Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014a. doi: 10.3115/v1/W14-4012.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014b. doi: 10.3115/v1/d14-1179.

G. Chrupala. Towards a machine-learning architecture for lexical functional grammar parsing. 05 2020.

A. Clifton and A. Sarkar. Combining morpheme-based machine translation with post-processing morpheme prediction. In D. Lin, Y. Matsumoto, and R. Mihalcea, editors, *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 32–42. The Association for Computer Linguistics, 2011.

R. Cotterell, C. Kirov, J. Sylak-Glassman, D. Yarowsky, J. Eisner, and M. Hulden. The SIGMORPHON 2016 shared task - morphological reinflection. In M. Elsner and S. Kübler, editors, *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, Berlin, Germany, August 11, 2016*, pages 10–22. Association for Computational Linguistics, 2016a. doi: 10.18653/v1/W16-2002.

R. Cotterell, H. Schütze, and J. Eisner. Morphological smoothing and extrapolation of word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin,*

Germany, Volume 1: Long Papers. The Association for Computer Linguistics, 2016b. doi: 10.18653/v1/p16-1156.

R. Cotterell, C. Kirov, J. Sylak-Glassman, G. Walther, E. Vylomova, P. Xia, M. Faruqui, S. Kübler, D. Yarowsky, J. Eisner, and M. Hulden. Conll-sigmorphon 2017 shared task: Universal morphological reinflection in 52 languages. In M. Hulden, editor, *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection, Vancouver, BC, Canada, August 3-4, 2017*, pages 1–30. Association for Computational Linguistics, 2017a. doi: 10.18653/v1/K17-2001.

R. Cotterell, J. Sylak-Glassman, and C. Kirov. Neural graphical models over strings for principal parts morphological paradigm completion. In M. Lapata, P. Blunsom, and A. Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pages 759–765. Association for Computational Linguistics, 2017b. doi: 10.18653/v1/e17-2120.

R. Cotterell, C. Kirov, J. Sylak-Glassman, G. Walther, E. Vylomova, A. D. McCarthy, K. Kann, S. J. Mielke, G. Nicolai, M. Silfverberg, D. Yarowsky, J. Eisner, and M. Hulden. The conll-sigmorphon 2018 shared task: Universal morphological reinflection. In M. Hulden and R. Cotterell, editors, *Proceedings of the CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, October 31 - November 1, 2018*, pages 1–27. Association for Computational Linguistics, 2018. doi: 10.18653/v1/k18-3001.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Do-

ran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/v1/n19-1423.

M. Dreyer and J. Eisner. Graphical models over multiple strings. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, EMNLP 2009, 6-7 August 2009, Singapore, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 101–110. ACL, 2009.

M. Dreyer, J. Smith, and J. Eisner. Latent-variable modeling of string transductions with finite-state methods. In *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1080–1089. ACL, 2008.

D. Eck and J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical report, 2002.

A. Erdmann, M. Elsner, S. Wu, R. Cotterell, and N. Habash. The paradigm discovery problem. *CoRR*, abs/2005.01630, 2020.

M. Faruqui, Y. Tsvetkov, G. Neubig, and C. Dyer. Morphological inflection generation using character sequence to sequence learning. In K. Knight, A. Nenkova, and O. Rambow, editors, *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*,

pages 634–643. The Association for Computational Linguistics, 2016. doi: 10.18653/v1/n16-1077.

A. M. Fraser, M. Weller, A. Cahill, and F. Cap. Modeling inflection and word-formation in SMT. In W. Daelemans, M. Lapata, and L. Màrquez, editors, *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*, pages 664–674. The Association for Computer Linguistics, 2012.

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks : the official journal of the International Neural Network Society*, 18:602–10, 07 2005. doi: 10.1016/j.neunet.2005.06.042.

H. Hassan, A. Negm, M. Zahran, and O. Saavedra. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. *International Water Technology Journal*, 5, 12 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

M. Hulden, M. Forsberg, and M. Ahlberg. Semi-supervised learning of morphological paradigms and lexicons. In G. Bouma and Y. Parmentier, editors, *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, April 26-30, 2014, Gothenburg, Sweden*, pages 569–578. The Association for Computer Linguistics, 2014. doi: 10.3115/v1/e14-1060.

K. Janecki. *300 Polish Verbs*. Barron's Educational Series, 2000.

H. Jin, L. Cai, Y. Peng, C. Xia, A. D. McCarthy, and K. Kann. Unsupervised morphological paradigm completion. *CoRR*, abs/2005.00970, 2020.

K. Kann and H. Schütze. Single-model encoder-decoder with explicit morphological representation for reinflection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016a.

K. Kann and H. Schütze. MED: the LMU system for the SIGMORPHON 2016 shared task on morphological reinflection. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, Berlin, Germany, August 11, 2016*, pages 62–70, 2016b. doi: 10.18653/v1/W16-2010.

A. E. Kibrik. The handbook of morphology. In A. Spencer and A. M. Zwicky, editors, *Andrew Spencer and Arnold M. Zwicky, editors*, pages 455–476. Oxford: Blackwell Publishers, 1998.

P. Makarov and S. Clematide. UZH at conll-sigmorphon 2018 shared task on universal morphological reinflection. In M. Hulden and R. Cotterell, editors, *Proceedings of the CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, October 31 - November 1, 2018*, pages 69–75. Association for Computational Linguistics, 2018. doi: 10.18653/v1/k18-3008.

P. Makarov, T. Ruzsics, and S. Clematide. Align and copy: UZH at SIGMORPHON 2017 shared task for morphological reinflection. *CoRR*, abs/1707.01355, 2017.

A. D. McCarthy, E. Vylomova, S. Wu, C. Malaviya, L. Wolf-Sonkin, G. Nicolai, C. Kirov, M. Silfverberg, S. J. Mielke, J. Heinz, R. Cotterell, and M. Hulden. The SIGMORPHON 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 229–244, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4226.

E. Minkov, K. Toutanova, and H. Suzuki. Generating complex morphology for machine translation. In J. A. Carroll, A. van den Bosch, and A. Zaenen, editors, *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics, 2007.

A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

K. Oflazer, Ö. Çetinoğlu, and B. Say. Integrating morphology with multi-word expression processing in Turkish. In *Proceedings of the Workshop on Multiword Expressions: Integrating Processing*, pages 64–71, Barcelona, Spain, July 2004. Association for Computational Linguistics.

F. Rosenblatt. *The Perceptron—a perceiving and recognizing automaton*. Report 85-460-1. Cornell Aeronautical Laboratory., 1957.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with

pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083, 2017. doi: 10.18653/v1/P17-1099.

W. Seeker and Ö. Çetinoglu. A graph-based lattice dependency parser for joint morphological segmentation and syntactic analysis. *Trans. Assoc. Comput. Linguistics*, 3:359–373, 2015.

A. Sharma, G. Katrapati, and D. M. Sharma. IIT(BHU)-IIITH at conll-sigmorphon 2018 shared task on universal morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection, Brussels, October 31 - November 1, 2018*, pages 105–111, 2018.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. E. Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28:*

Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, pages 2773–2781, 2015.

E. Vylomova, J. White, E. Salesky, S. J. Mielke, S. Wu, E. Ponti, R. H. Maudslay, R. Zmigrod, J. Valvoda, S. Toldova, F. Tyers, E. Klyachko, I. Yegorov, N. Krizhanovsky, P. Czarnowska, I. Nikkarinen, A. Krizhanovsky, T. Pimentel, L. T. Hennigen, C. Kirov, G. Nicolai, A. Williams, A. Anastasopoulos, H. Cruz, E. Chodroff, R. Cotterell, M. Silfverberg, and M. Hulden. The SIGMORPHON 2020 Shared Task 0: Typologically diverse morphological inflection. In *Proceedings of the 17th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, 2020.

S. Wu and R. Cotterell. Exact hard monotonic attention for character-level transduction. In A. Korhonen, D. R. Traum, and L. Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 1530–1537. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1148.

W. Zhao, L. Wang, K. Shen, R. Jia, and J. Liu. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 156–165, 2019.