University of Helsinki
Faculty of Arts
Language Technology

Bachelor's thesis

Using Ensemble Methods to Improve the Accuracy of Morphological Inflection RNNs

Ilmari Kylliäinen
014596801

Supervisors:
Miikka Silfverberg
Jörg Tiedemann

June 9, 2019

Abstract

In order to create natural language processing systems of which use isn't restricted to only languages with more simple morphology, it is important to develop systems that focus on modeling morphology. This task is important especially when it comes to languages with complex morphology because their words may have even hundreds of possible forms. A great number of word forms makes it more difficult to gather high-quality statistics which leads to data sparsity and degrades the systems' performance. Different machine learning methods for modeling morphology and especially word inflection have been found effective for this problem.

This thesis examines the performance of two model-combining ensemble methods, bagging and majority voting, on neural network models designed for morphological inflection. The results demonstrate that using ensemble methods is a viable strategy for improving the performance of artificial neural networks designed for the inflection task. Of the two compared methods, majority voting obtains better results for almost every tested language and training data setting, but also bagging ensembles mostly outperform individual models.

Abstract

Jotta korkeatasoisia luonnollisen kielen käsittelyn sovelluksia voitaisiin luoda myös morfologialtaan monimutkaisemmille kielille, on tärkeää kehittää sovelluksia, joiden avulla morfologiaa pystytään mallintamaan. Tehtävä on erityisen tärkeä kielille, joiden sanoilla on useita taivutusmuotoja, sillä taivutusmuotojen suuri lukumäärä vaikeuttaa korkealaatuisen statistiikan keräämistä, mikä taas johtaa kieliaineiston harvuuteen ja huonompaan suorituskykyyn. Koneoppimisalgoritmeihin pohjautuvat lähestymistavat morfologian ja etenkin sanojen taivutuksen mallintamiseen ovat osoittautuneet hyvin toimiviksi ratkaisuiksi tähän ongelmaan.

Tämä kandidaatintutkielma tutkii mahdollisuuksia parantaa sanantaivutusneuroverkkomallien suorituskykyä käyttämällä useita koneoppimismalleja yhdistäviä ensemble-menetelmiä. Vertailen kahden ensemble-menetelmän, baggingin ja majority votingin, tehokkuutta sanojen taivutukseen suunnitelluilla neuroverkkomalleilla. Tulokset osoittavat, että ensemble-menetelmien hyödyntäminen sanantaivukseen suunniteltujen neuroverkkojen kanssa on käyttökelpoinen strategia. Vertailluista menetelmistä majority voting tuottaa parhaita tuloksia lähes jokaisella kielellä ja aineistokoolla, mutta myös baggingin tulokset ovat pääosin parempia kuin yksittäisillä malleilla saadut tulokset.

Contents

Acknowledgements

First, I would like to thank my supervisors Miikka Silfverberg and Jörg Tiedemann for giving me valuable guidance and feedback throughout the study and helping me structure and finish this thesis. Second, I would also like to thank everyone who has shown interest in my thesis and thereby forced me to reflect and put my work into words by having to explain it. I also wish to acknowledge CSC – IT Center for Science, Finland, for computational resources.

1 Introduction

1.1 Context

Natural language processing (NLP) systems that perform tasks such as machine translation, speech recognition, information retrieval and question answering typically consist of several subsystems that analyze text/voice input from different perspectives. These subsystems often perform tasks, like *segmentation* (separating chapters and sentences), *tokenization* (division to semantic units, often words), *POS tagging* (assigning parts of speech to each token) and *parsing* (dividing the text into different components and defining their syntactic and/or semantic relations). For languages with large speaker communities such as English, these subsystems do exist and can be used to build full-fledged NLP applications. The situation is very different when it comes to languages which have much smaller data resources or more complex morphology.

Syntactic parsing is a very complex task because of the ambiguous nature of natural language. For morphologically complex languages, this task is even more difficult because words may contain several affixes and these languages often have very rich grammatical case systems. The *data sparsity* caused by highly inflected words degrades the performance of otherwise state-of-art systems in standard tasks, such as POS tagging (Kondratyuk et al., 2018) and parsing (Ballesteros et al., 2015). Sparsity doesn't mean that the data itself is of worse quality but rather insufficient. It refers to a phenomenon in which the text data used as the training data when creating, for example, a machine translation system, can contain only a portion of all possible word forms due to the large number of possible word forms. Gathering high-quality statistics becomes more difficult because most n-grams of complete word forms are rare, and thus the systems are unable to model languages accurately enough.

In order to create systems of which use isn't restricted to only languages with more simple morphology, it is important to develop NLP systems that focus on modeling morphology. Downstream NLP applications must be able to both generate and analyze morphological forms. Creating systems that are able to accomplish these tasks is one way to deal with the data sparsity caused by a complex morphological system. *Morphological inflection* (MI) is a task in which the object is to generate an inflected word form using a lemma and a morphosyntactic description. For example, if the input is the Italian lemma *mangiare* 'to eat' and the morphosyntactic description is **V;IND;FUT;1;SG**, the output should be the 1st person singular future in-

dicative form *mangerò*. In the inverse direction, when analyzing words, the input should be an inflected form and the output should be a lemma and morphosyntactic description (e.g. *mangerò* → *mangiare* + **V;IND;FUT;1;SG**). Another prominent and more challenging task is *morphological reinflection* (MRI) where an inflected form is generated from another inflected form which may not be the lemma.

Some fundamental morphology-related terminology I use in this thesis are *word form*, *lemma*, *inflected form*, *POS* and *morphosyntactic description*. Words forms are all the different ways a word can exist in the context of a language. Lemma refers to an uninflected basic form, and inflected forms are all the other word forms of a word than the lemma. POS - or part of speech - is a particular grammatical class of a word which is assigned in accordance with its syntactic functions. For example, noun, adjective, or verb. Morphosyntactic description contains features that are related to an inflected form. The features can be, for example, gender, number, person, case or aspect.

1.2 Purpose

During recent years, different machine learning methods have been found effective for different morphology tasks (Mahmoudi et al., 2013; Creutz and Lagus, 2007; Kann et al., 2016), and morphological inflection in particular. Especially implementations based on one of its branches, deep learning, have obtained great results (Cotterell et al., 2017, 2018). In deep learning, an Artificial Neural Network (ANN) consisting of several layers is trained on large amounts of data in order for it to learn to make its own predictions. In this thesis, I will group ANNs to *Shallow* and *Deep* neural networks. Shallow neural networks are NNs that usually have from zero to two hidden layers as opposed to deep NNs which have several of them, often of various types. Whereas all machine learning can be characterized as learning to make predictions based on past observations, deep learning systems independently learn representations for the data which facilitates the prediction task. In the case of morphological inflection, the systems aim to yield an abstract representation of the word inflection process and thereby to learn how to inflect words. Some fundamental information on the basic structure and functionality of ANNs will be given in Section 2.1.

The use of deep learning methods is still a rather new approach for MI and MRI tasks. Regardless of the promising results, the use of so-called *ensemble methods* to improve the performance of systems designed for these tasks is

yet to be thoroughly investigated. Ensemble learning is the process by which multiple machine learning models are combined to achieve better prediction performance. Despite abundant research on the theoretical properties of ensemble learning and the methods' performance in general, their behavior with deep networks is still not well understood. While deep learning has been successfully applied to a variety of NLP tasks, including machine translation, semantic segmentation, and morphological generation tasks, few studies have fully investigated ensembles of deep ANNs, especially on *sequence-to-sequence* (seq2seq) predictors, like the ones used for MI and MRI. This thesis compares the performance of two quite similar ensemble methods, *bagging* and *majority voting*, on MI neural network models. Since they are both techniques that aim to reduce model variance, and the architecture used in the thesis's experiments, so-called *encoder-decoder*, is known to obtain high variance (Denkowski and Neubig, 2017), it is reasonable to presume that using them could lead to better results than an individual model could achieve.

1.3 Outline

The thesis consists of 6 chapters. Chapter 2 goes through earlier research on computational approaches to MI task and the use of ensemble methods in deep learning. It also provides some fundamental information on ensemble methods and ANNs, and an introduction to relevant terminology. Chapter 3 describes the key components of the used neural network architecture and its implementation and explains bagging in more detail. Chapter 4 introduces the CoNLL-SIGMORPHON 2017 data considered in this study. Baseline models, ensembles and all the results obtained from the experiments are presented in Chapter 5. Finally, Chapter 6 discusses the results, summarises the work done, and draws some conclusions.

2 Related Work

2.1 Morphological Inflection

The field of computational morphology has attracted considerable attention throughout the history of language technology and many methods have been applied to problems in morphology. These include both machine learning models and models that exploit the paradigm structure of language. The traditional approach to MI is to create a *lexicon* (word list) with high coverage

of lemmas of a language and a *finite-state transducer* (FST), for example, a so-called two-level morphology model (Koskenniemi, 1984) that is designed to model the desired language. Finite-state transducers are rule-based systems that accept a predefined set of inputs and can produce a matching output. Thus, in the case of morphological inflection, the transducers map between lemmas and inflected forms following language-specific, handcrafted rules. Carefully designed and robust FSTs are time and space efficient and provide the capability to generate and analyze well-formed words in a language. However, building FSTs requires deep knowledge of the target language and developing and maintaining such systems can be very time-consuming and expensive (Wintner, 2008). Some other downsides with FSTs are that they are generally unable to analyze and generate any novel lexical items and spelling errors also might cause problems. Having a large FST system that covers the majority of a language's vocabulary can also lead to very long compilation times. This can be problematic because doing even minor changes requires usually at least a partial compilation.

The neural word inflection system by Rumelhart and McClelland (1986) was the first attempt to model word inflection using artificial neural networks. The system was designed to inflect English verb stems to their past forms. As mentioned in Section 1.2, an ANN consists of several layers. The layers, in turn, consist of several *neurons* that are connected to and receive inputs from neurons in preceding layers. Each neuron has a *weight* associated with each of its connection, and the weights are constantly being updated during a model's training. The input layer passes the information forward to the next layer, and no computations are performed in any of its neurons. The input is multiplied by the weights of the possible hidden layers, summed, and then passed through a non-linear *activation function* of choice (e.g. sigmoid, ReLU, tanh). The hidden layers apply a non-linearity to their inputs, and the more hidden layers are stacked together, the more complex functions it is possible for the network to model. After all the data processing in the previous layers is done, the output layer of the ANN transmits the output information accordingly in the way it has been designed to give. The ANN introduced by Rumelhart and McClelland produced correct past tense forms for the training verbs and it was also able to generate correct forms for some unfamiliar verbs.

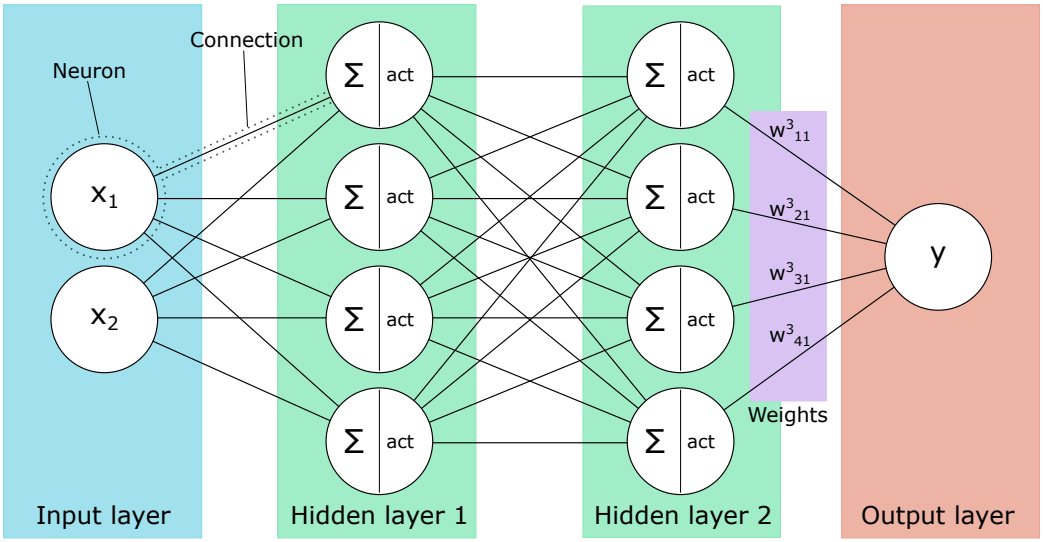

Figure 1: Example of an ANN with 2 hidden layers.

Compared to contemporary neural networks, the system by Rumelhart and McClelland (1986) was very simple. It consisted of just two layers, input and output, which both had 460 neurons. The weights for the connections had to be set manually by the network's builder. Rumelhart and McClelland concluded that their network had reproduced human learning as the model displayed a number of phenomena known to characterize children's behavior, like overregularization of irregular verbs. In general, the system made many errors in computing the past tense forms of a large percentage of the words it was tested on, and it failed to generate any past tense form at all for certain words. The many flaws of the system along with their claims on neural networks in general were highly criticized by Pinker and Prince (1988). This led to many linguists and cognitive scientists to not consider neural networks a viable approach to modeling linguistic data and human cognition for years (Kirov and Cotterell, 2018). However, this mindset has undergone major changes in the field of NLP along with growing computational power, new machine learning methods, datasets, and particularly the introduction of RNN networks which were not available to Rumelhart and McClelland (1986).

Due to the limitations of FST approaches and the development of novel architectures, the interest in applying machine learning methods to also computational morphology related tasks has grown. Modeling morphology is considered an interesting and important research problem, and machine learning approaches have already proven to obtain great results in MI tasks. When

comparing to FSTs, one major benefit of machine learning approaches is their flexibility. The systems can learn to analyze and generate words forms and lemmas they have never seen. Such data-driven approaches to the task of MI have received a lot of attention lately due to several morphology-related shared tasks organized by CoNLL and SIGMORPHON. The three most recent shared tasks[1] consisted of MI tasks and additional tasks like MRI and inflection paradigm filling, depending on the year. For SIGMORPHON 2016 shared task on Morphological Reinflection (Cotterell et al., 2016), Kann and Schütze presented a language independent sequence-to-sequence attention encoder-decoder system, MED. The 2016 shared task consisted of both MI and MRI subtasks, and the architecture MED was designed to work with both. It scored first in all of the 90 subtasks of the shared task's final evaluation. Of the 11 submitted systems, the 5 neural systems were clear winners in the shared task. The MED system is based on the encoder-decoder architecture with attention proposed by Bahdanau et al. (2014), which is an extension of the recurrent neural network encoder-decoder system developed by Cho et al. (2014) and Sutskever et al. (2014). This encoder-decoder based model architecture was the most popular approach in the two subsequent CoNLL shared tasks (Cotterell et al., 2017, 2018) in which both of the most successful systems, Makarov et al. (2017); Makarov and Clematide (2018), used variants of the encoder-decoder approach. The structure of the architecture presented by Kann and Schütze (2016) will be more closely examined in the next chapter as models for baseline and experiments are trained with an implementation with similar architecture.

The recent work on neural approaches to modeling morphology has been focusing mainly on generating inflected forms, and not so much on analyzing them. None of the CoNLL-SIGMORPHON shared tasks have included word form analyzing tasks either. Morphological analyzing is, however, considered an important task because it could help reduce sparsity by providing several out-of-context morpheme-based analyses for words. Analyzing is also considered a bit more difficult task because analyzers usually introduce ambiguity by returning multiple analyses for the same inflected form which then requires another step of morphological disambiguation to choose the correct analysis in context. However, neural approaches to also morphological analyzing have obtained good results. For example, Moeller et al. (2018) with Arapaho verbs and Zalmout and Habash (2017) with Arabic.

[1]CoNLL-SIGMORPHON shared tasks 2016, 2017 and 2018

2.2 Ensemble Methods and Deep Learning

As stated in section 1.2, an ensemble consists of a set of individually trained models whose predictions are combined when classifying novel instances or generating sequences. This approach aims to the production of better predictive performance compared to a single model. The aim of any ensemble method is to reduce either a model's bias or variance. High bias causes the used algorithm to miss the relevant relationship between the input and the output variable. It means that the model can't capture the complexity of data and thus *underfits* it and doesn't perform well. High variance, in turn, means that the model is sensitive to random noise in the training data. When that occurs, the model performs well on the training dataset but doesn't do well with other sets, like validation or test set. The goal of any supervised machine learning algorithm is to simultaneously reduce errors caused by bias and variance.

Two widely used methods for creating ensembles are boosting (Schapire, 1990) and bagging (Breiman, 1996). When it comes to classification tasks, they are both quite well understood and it has been shown that taking an ensemble of models which were initialized differently and trained independently leads to improved performance (Hansen and Salamon, 1990; Collobert et al., 2011). Evaluation has been done especially on aggregation algorithms (Maclin and Opitz, 1999; Granitto et al., 2005). Aggregation methods use different techniques, like majority voting, to aggregate the outputs of several models trained in parallel. However, as far as the use of ensemble methods with ANNs in general is considered, the existing research focuses on classification tasks and shallow ANNs (Hansen and Salamon, 1990; Maclin and Opitz, 1999). Most of the neural network literature focuses mainly on the design of the network structure and only applies naive ensembling techniques, like averaging the outputs, to enhance the performance.

One so-called naive ensemble technique, to which I refer as *majority voting* in this thesis, is to simply train several individual models with the same training data and then combine their outputs to one final output. This technique further divides into naive majority voting (NMV) and weighted majority voting (WMV). In NMV, the most frequent output sequence is chosen as the final output, and in WMV, each model is assigned with a weight based on its accuracy[2] on a held-out development set. Majority voting is used to address the effects of random initialization which makes each individual ANN model perform a bit differently.

[2]For example, if a model's accuracy is 87%, its weight in voting is 0.87.

The naive majority voting technique was a popular choice for MI tasks in the CoNLL shared tasks. The MED system for SIGMORPHON 2016 was an ensemble of 5 models trained on the same training data but with different random parameter initializations. The systems with overall best performances in SIGMORPHON 2017 and 2018 by Makarov et al. (2017) and Makarov and Clematide (2018) were also ensembles trained on the same training data, but the number of models in the ensembles varied from 5 to 15 depending on the size of the available training data. The smaller the training data set was, the more models the ensembles consisted of. In SIGMORPHON 2018 some teams also combined multiple models with different architecture, for example, neural and non-neural. The previously mentioned systems demonstrated that ensembling models trained on the same training data can increase neural systems' performance. However, other methods for ensemble learning, such as boosting and bagging, were not used in the shared tasks, and no research exists on their use with neural word inflection models.

The two methods compared in this thesis, bagging and majority voting, both aim to reduce model variance and thereby prevent overfitting. However, their approach to doing so is different. Whereas bagging varies the training set, majority voting varies the model parameter initialization. That is, in bagging each ensemble member is trained on different training data and model parameter initialization, and in majority voting, the members are trained on the same training data and with different initialization. Like majority voting, bagging is also based on the idea that outputs generated by several models in the ensemble are more likely to prove correct than outputs generated by fewer models. Bagging will be more closely discussed in Section 3.3.

3 Models and Methods

3.1 RNN Architecture

It is very typical that the input and output data for NLP systems are various types of sequences, for example, sequences of words, morphs or characters. Tasks with sequential input and/or output include machine translation, sentimental analysis, morphological inflection and speech recognition, to mention a few.

Source sequence	**Length**	**Target sequence**	**Length**
írása N;TERM;SG	8	írásáig	7
converger V.PTCP;PST;FEM;PL	13	convergidas	11

Table 1: Examples of source and target sequences in an MI task. Each character and morphosyntactic tag is of length 1.

More traditional neural networks, such as *feedforward networks* (FFNN) don't perform very well with arbitrarily sized input sequences. FFNNs are called feedforward because the input to the network turns into output after flowing through intermediate computations with weights and there are no feedback connections that would output the network's output back into itself. The input for a classical FFNN needs to be a fixed size vector and there is no immediately obvious way of encoding an arbitrary sequence into a fixed size vector. This is problematic especially if information about the order of the elements needs to be preserved. For some tasks it is possible to use unordered sequence representations, like a *continuous bag of words* (CBOW), and vector concatenation and/or addition with FFNNs to encode arbitrary length sequences as fixed sized vectors. However, also this approach disregards the order of the input sequence parts, and it obviously doesn't lead to very good results if the system cannot distinguish between input lemmas where the character order changes, like "skin" and "sink".

Recurrent neural networks (RNN, Elman, 1990) are designed to model arbitrarily sized input sequences while preserving the order of features. RNNs have a built-in feedback loop that allows them to work as forecast engines. They work on the principle of saving the output, *hidden state*, of a layer and feeding it back to the input in order to predict the output of the layer. The information is looped in every layer, and each RNN run is called a *time step*. RNNs have the capability to memorize the inputs due to their internal memory. Key idea is that each timestep has a layer with the same weights,

and each time step takes the input at the current time step as well as the hidden state from the previous time step. RNNs are trained by *unrolling* them into very deep feedforward networks, where each time step of an input sequence corresponds to a new layer processed by the network.

3.1.1 Encoder-Decoder

Encoder-decoder structure was first introduced in 1997 when Neco and Forcada came up with the idea of using it to do machine translations. It has since developed to be an effective and standard approach for MI, neural machine translation (NMT) and seq2seq tasks in general. The architecture is currently also utilized by Google's NMT system. Encoder-decoder RNNs have two key components: an *encoder* network and a *decoder* network.

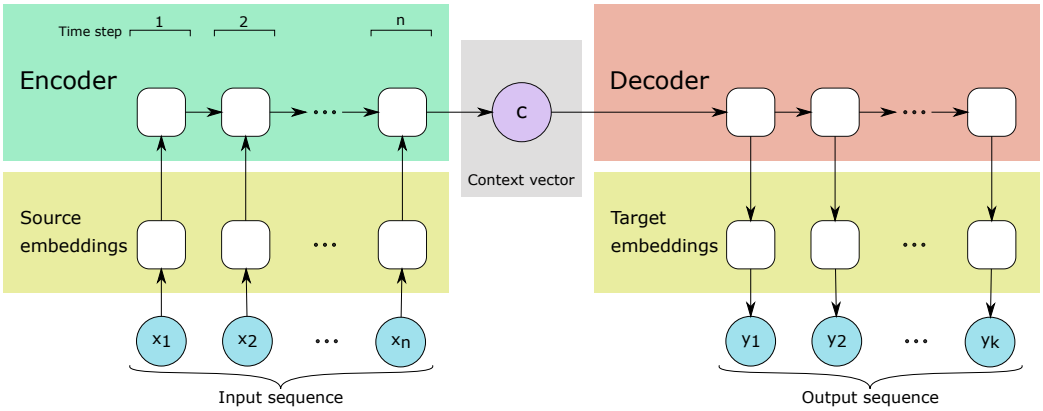

Figure 2: Encoder-decoder architecture.

First, the input layer takes the input sequence as input and passes it to the embedding layer which converts sequence parts to fixed-length vectors, *embeddings*. Embeddings are low-dimensional vector representations of a vocabulary (in the case of MI, all the characters and morphosyntactic tags), of which contextual similarities they preserve. The encoder network then takes the embeddings as input and encodes them into a fixed-length internal representation, a *context vector*. Encoding the whole input sequence into one vector has its weaknesses which will be further discussed in Section 3.1.3. A bidirectional encoder, like the one used in Kann and Schütze (2016) involves duplicating the first recurrent layer in the network so that there are two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second. This causes each hidden state at each time step not to only depend on the

preceding, but also on the following input. The output of the encoder is used as input to another network and not for prediction. In seq2seq models, the encoder is connected to the decoder network which uses the context vector as a "seed" from which to generate an output sequence.

3.1.2 LSTM

During the training of a deep neural network, a *loss function* is used to measure the inconsistency between the model's prediction and the correct output. In each iteration of training, each of the neural network's weights receives an update proportional to the partial derivative, *gradient*, of the loss function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. Vanishing gradients are problematic especially with RNNs because they involve a lot of multiplication due to the unrolling. In consequence, the training process takes very long and the prediction accuracy of the model will decrease.

One solution to this *vanishing gradient problem* is to use a technique called *gating*. Gating helps the network to decide when to forget the current input and when to remember it for future time steps. One popular gating architecture is *Long Short Term Memory* (LSTM, Hochreiter and Schmidhuber, 1997). The LSTM *cells* are used as building units for the layers of an RNN, which is often called an LSTM network. They are a special kind of RNNs that are capable of remembering information for long periods of time. An LSTM cell takes a hidden state and a cell state as input and then through intermediate computations passes them to the next cell. A cell has three internal mechanisms called *input gate*, *output gate* and *forget gate*. The gates regulate the flow of information by learning which part of the input sequence is important and which can be forgotten. LSTMs solve the vanishing gradient problem by creating a connection between the forget gate activations and the gradients computation. Whereas regular RNNs apply multiplicative updates to the hidden state of the network, LSTMs apply additive updates. This is what counteracts the vanishing gradient problem and leads to the network being able to use relevant information to make predictions.

3.1.3 Attention

Even though LSTM partly removes the vanishing gradient problem, it is clear that a fixed size vector cannot encode an arbitrarily long sequence. This means that the traditional encoder-decoder architecture is bound to forget features from the input string when the input size grows. *Attention* was first introduced in 2014 when Bahdanau et al. used one in a neural machine translation system. They have since shown to produce state-of-the-art results in machine translation and many other NLP tasks. In order to avoid encoding all the information into just one vector, attention mechanisms allow the decoder to "attend" different parts of the input sequence. The attention model develops a context vector that is filtered specifically for each output time step and adds a search over the input sequence to get the positions where the relatively most relevant information for this particular output sequence position is concentrated. In other words, context vectors are computed for every part of the output sequence during encoding. With this source position information and the previously generated parts of the output sequence, the model predicts a new part to the output. Attention mechanisms were used in all the best-performing systems submitted to the last three SIGMORPHON shared tasks.

3.2 Implementation

The LSTM attention encoder-decoder models for baseline and experiments are implemented using the OpenNMT neural machine translation toolkit (Klein et al., 2017). The character embeddings for input and output characters are 100-dimensional. The embeddings are processed by a 1-layer bidirectional LSTM encoder (BRNN) with hidden state size 300 which consists of two independent encoders: one encoding the normal sequence and the other the reversed sequence. The encoder representations are then fed into a 1-layer LSTM attention decoder with hidden state size 300.

When training[3], a batch size of 64 is used and all the models are trained for 12,500 training steps where one step corresponds to updating on a single batch. Model parameters are optimized using the Adam optimization algorithm (Kingma and Ba, 2014) with learning rate of 0.001.

[3]All the scripts used to train models, form ensembles and find out their accuracies are available in https://github.com/ilmarikyl/ba-thesis.

3.3 Bagging

Bagging - which refers to *bootstrap aggregation* - is a widely used ensemble method for generating multiple versions of a predictor and combining these to get an aggregated predictor. Bootstrapping refers to random sampling with replacement (meaning that an example can be selected multiple times). Bagging was first introduced by Breiman (1996), and it was designed for use with models that have low bias and high variance, which makes it a promising technique for deep neural networks.

A quite standard way to create a bagging ensemble is to draw with replacement M resamples L_n $(n = 1, M)$ from data set D of size N. In consequence, there are on average $0.63N$ different examples in each resample (Efron and Tibshirani, 1993). Each resample L_n is then used to train M independent models. After each model is trained in parallel with the resamples as training data, a prediction is given based on the aggregation of predictions from all the models. Depending on the task, the aggregation can be, for example, an averaged or majority voted output of all the models in the ensemble. In the case of MI or any other sequence generating task, the final output can simply be voted.

The goal is to reduce the variance by averaging the models together without significantly increasing the bias. In other words, bagging prevents overfitting. Taking a sample of the data allows the resample to contain different characteristics than it might have contained as a whole. The examples that don't get picked for resamples can be used for validation purposes if there isn't a separate validation set. There is no thorough research on the optimal number of models in a bagging ensemble. For classification tasks, it has been shown that the optimal ensemble sizes show great variation for different problems (Hernández-Lobato et al., 2013). From the perspective of introducing training data diversity to the ensemble, a number of models as large as possible is advisable.

4 Data

Language data for 10 different languages in CoNLL-SIGMORPHON 2017 Task 1 dataset is used to train and evaluate the baseline models and ensembles. The chosen languages are Arabic (ARA), Finnish (FIN), Georgian (GEO), German (GER), Hindi (HIN), Italian (ITA), Khaling (KHA), Navajo (NAV), Russian (RUS) and Turkish (TUR). The language set is diverse in terms of morphological structure, and the languages also encompass different morphological properties and inflection processes. There are languages which use primarily prefixes (Navajo), suffixes (Turkish) or both infixes and suffixes (Georgian). The languages also exhibit features such as vowel harmony (Turkish, Finnish), consonant harmony (Navajo), and templatic morphology (Arabic). For 9 of the 10 languages, the data is gathered from the English edition of Wiktionary, a dictionary containing morphological paradigms for several lemmata. Data for Khaling was created as part of the Alexina project (Walther et al., 2013). The language data for each language consists of training sets of three sizes (high, medium and low), a development set and a test set.

	High	**Medium**	**Low**
Training	10,000	1,000	100
Development		1,000	
Test		1,000	

Table 2: Data set sizes per language.

The data used in the experiments consists of tab separated files with three columns: lemma, inflected form, and morphosyntactic description[4]. The training data sets are sparse in the sense that they include only a few inflected forms for each lemma instead of complete inflectional paradigms.

[4]For example, `überbewerten überbewerteten V;IND;PST;3;PL`

5 Experiments

5.1 Baseline models

To compare the results of bagging and majority voting ensembles, a baseline must be established. For each of the 10 selected languages, 10 models were trained with differently initialized weights. This was done with both high and medium training data setting. The low training data was not used in experiments because initial testing resulted in very low accuracies. The accuracies were found out using held-out test sets. According to preliminary experiments, the development accuracy and perplexity[5] for each language converged around 6,000-10,000 training steps for each dataset. One training step corresponds to updating on a single batch (64 examples).

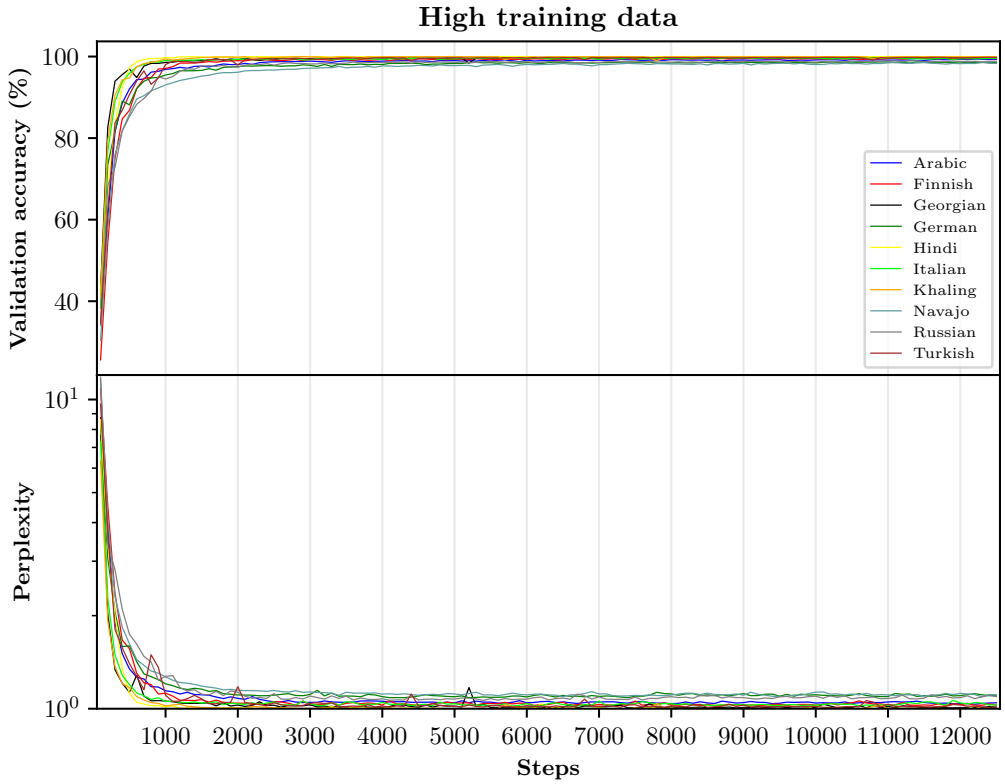

Figure 3: Baseline model validation accuracy and perplexity for each language during training with high training data.

[5]Model's perplexity is its degree of uncertainty when predicting the inflected form.

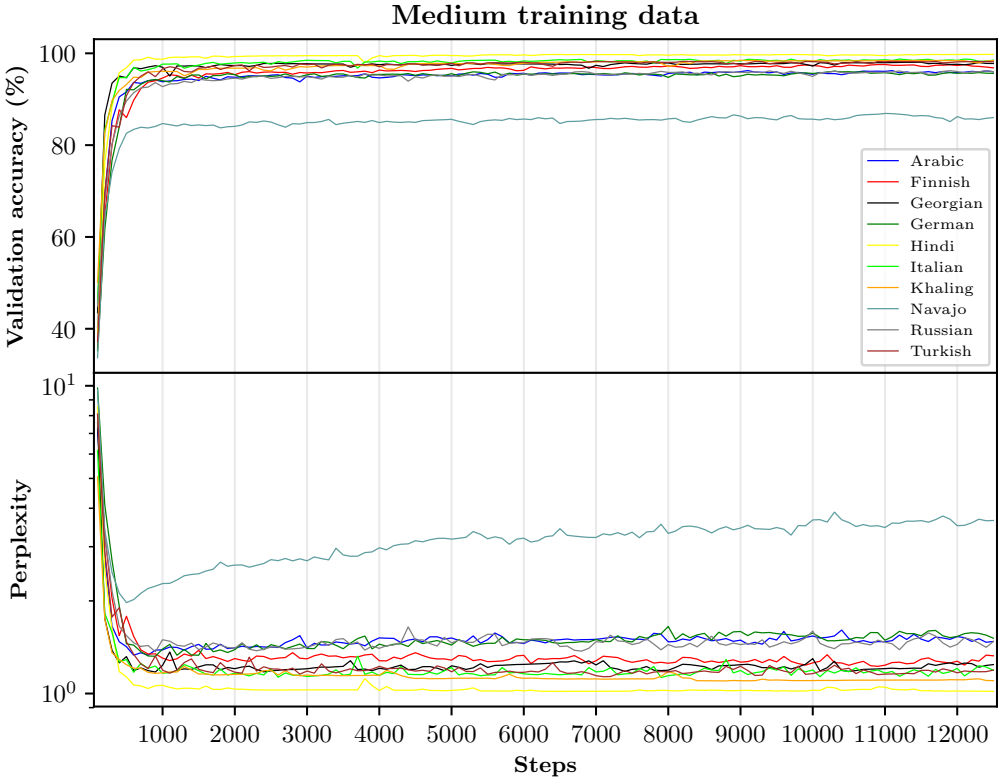

Figure 4: Baseline model validation accuracy and perplexity for each language during training with medium training data.

As can be seen in Figures 3 and 4, the training process for all the languages seems quite similar on both, high and medium training data settings. Almost all the languages' validation accuracies and perplexities converge before 10,000 steps with both data set sizes. One exception when training with medium data is Navajo which has significantly worse validation accuracy during the whole training and of which perplexity starts growing after about 500 training steps. This might be due to quick overfitting or Navajo's highly complex morphology. It is also worth mentioning that because the same training step amount and batch size are used on both high and medium training settings, the amount of training epochs[6] is much higher on medium setting which might also partly explain the possible overfitting problem with Navajo.

[6]One epoch is when the entire dataset is passed both forward and backward through the neural network only once.

5.2 Majority voting ensembles

The 10 models of each language were used to form also majority voting ensembles. Both naive majority voting and weighted majority voting were applied to get final output forms for the two ensembles per language. In NMV, the most frequent output sequence was chosen as the final output. In WMV, each model was assigned with a weight based on its own accuracy. The weights of the same predictions were then summed up and the prediction with the highest score was chosen as the final output. In case of a tie, the answer was chosen randomly among the most frequent predictions.

Figures 5 and 6 illustrate the baseline model and majority voting ensemble accuracies. As with the best-performing systems submitted to the CoNLL shared tasks, an improvement in accuracy can be seen when using naive or weighted majority voting. Compared to the baseline models, both majority voting ensemble types have statistically significantly better performance at the 95% confidence level as measured by a two-sided t-test. For both data set settings, using weighted majority vote to combine the outputs seems to result in even better accuracies for most languages.

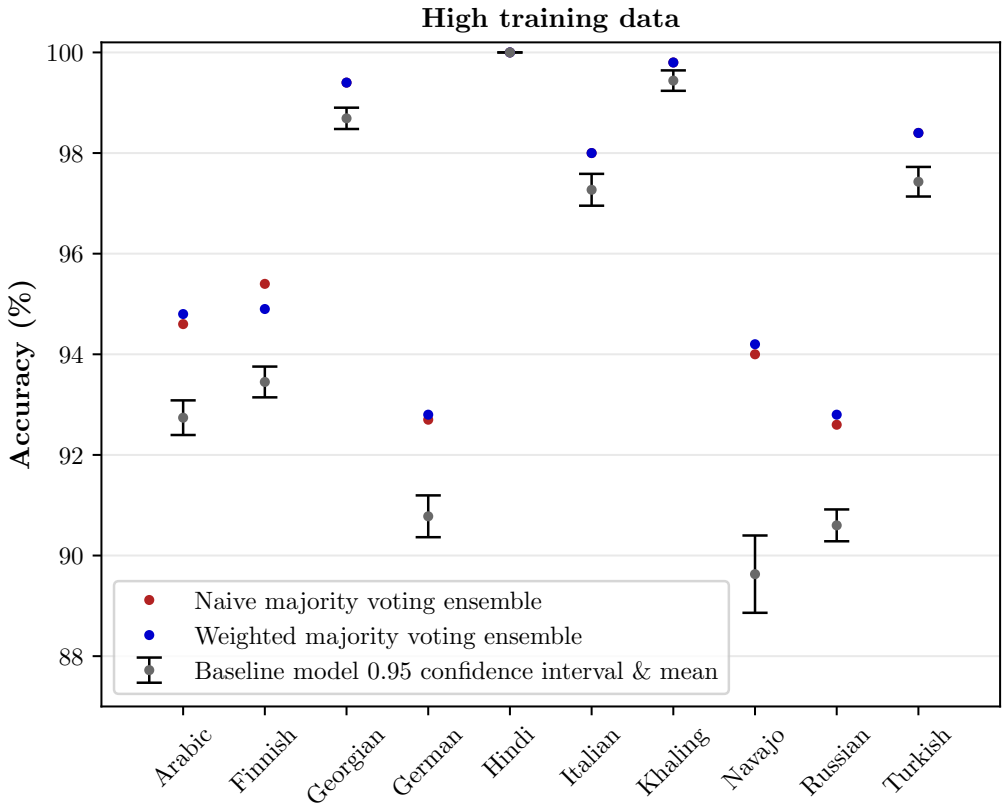

Figure 5: Accuracies of baseline models and majority voting ensembles on high setting.

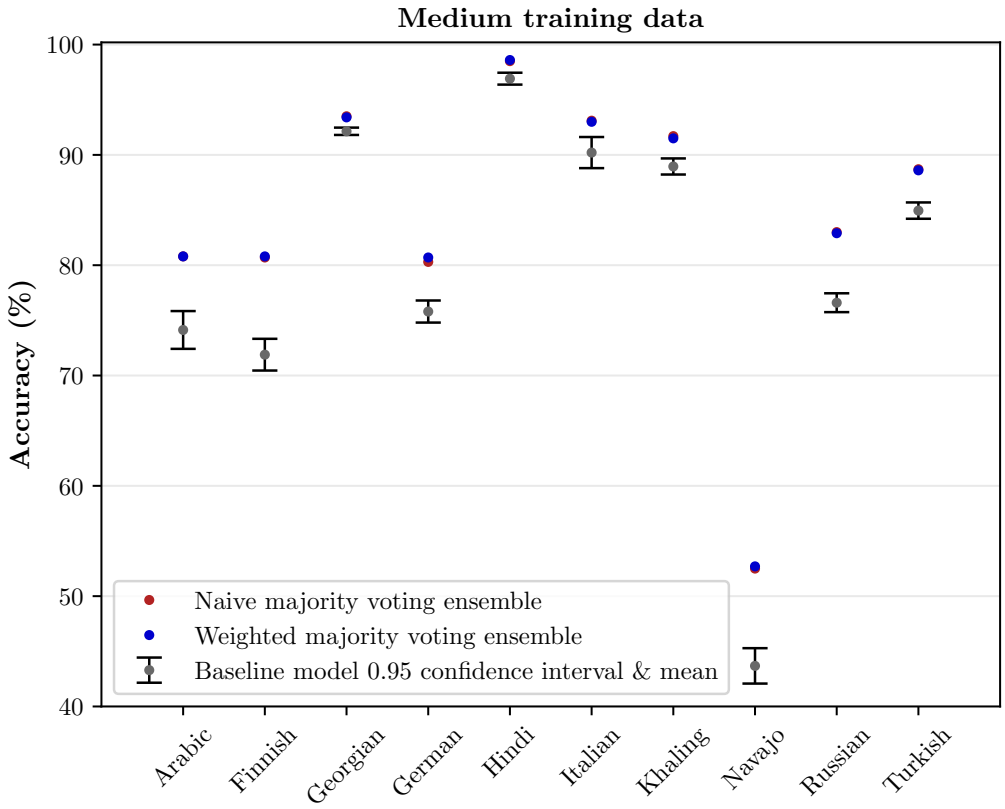

Figure 6: Accuracies of baseline models and majority voting ensembles on medium setting.

5.3 Bagging ensembles

5.3.1 Experimental setup

In total, four bagging experiments are conducted. In experiments 1 and 2, bagging ensembles of 10 and 100 models are formed by resampling from the high training set. In experiments 3 and 4, the ensembles consist of 10 and 100 models as well, but they are resampled from the medium training data. The resamples are chosen randomly with replacement and they are the same size as the data sets they are picked from.

Exp #	Dataset size	Resample size	Models in ensemble
1	10,000	10,000	10
2	10,000	10,000	100
3	1,000	1,000	10
4	1,000	1,000	100

Table 3: Sizes of datasets and resamples and number of models in each ensemble.

Each ensemble member is trained with different resample as training data. In addition to using different data for training, diversity between the ensemble members is ensured also by initializing their parameters differently. In order to evaluate the ensembles' performance, a held-out test data set is then fed as input. Both naive majority voting and weighted majority voting are applied to outputs of each model to form two ensembles for each language. In case of a tie in voting, the final output is chosen randomly among the most frequent predictions. As with the baseline models and MV ensembles, each bagging ensemble's final outputs are then compared to the "gold" predictions of the test data sets to compute the exact-match accuracies.

5.3.2 Results

Bagging results are illustrated in Figures 7 and 8. Table 4 shows the performance of all the ensembles as well as the baseline models' means and accuracies of best-performing baseline models.

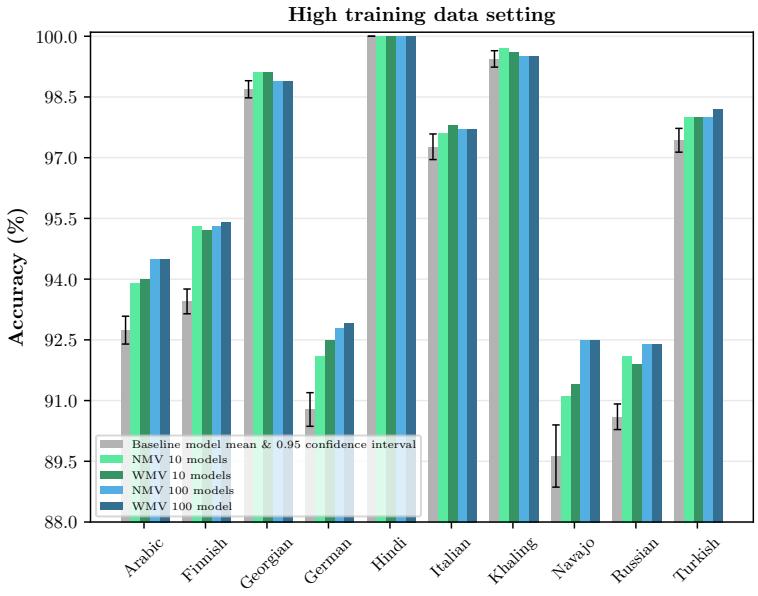

Figure 7: Experiments 1 & 2. Accuracies of bagging ensembles trained on the high training data and resample size of 10,000.

Figure 8: Experiments 3 & 4. Accuracies of bagging ensembles trained on the medium training data and resample size of 1,000.

		ARA	FIN	GEO	GER	HIN	ITA	KHA	NAV	RUS	TUR
	Best baseline model	93.40	94.00	99.10	91.60	**100.00**	**98.00**	**99.90**	91.30	91.50	98.00
	Baseline mean	92.74	93.45	98.69	90.78	**100.00**	97.27	99.44	89.63	90.60	97.43
	MV 10.NMV	94.60	**95.40**	**99.40**	92.70	**100.00**	**98.00**	99.80	94.00	92.60	**98.40**
	MV 10.WMV	**94.80**	94.90	**99.40**	92.80	**100.00**	**98.00**	99.80	**94.20**	**92.80**	**98.40**
High	Bagging 10.NMV	93.90	95.30	99.10	92.10	**100.00**	97.60	99.70	91.10	92.10	98.00
	Bagging 10.WMV	94.00	95.20	99.10	92.50	**100.00**	97.80	99.60	91.40	91.90	98.00
	Bagging 100.NMV	94.50	95.30	98.90	92.80	**100.00**	97.70	99.50	92.50	92.40	98.00
	Bagging 100.WMV	94.50	**95.40**	98.90	**92.90**	**100.00**	97.70	99.50	92.50	92.40	98.20
	Best baseline model	76.80	75.60	92.50	78.60	98.10	92.10	90.00	47.30	78.00	86.90
	Baseline mean	74.13	71.89	92.14	75.80	96.91	90.21	88.95	43.68	76.60	84.95
	MV 10.NMV	**80.80**	80.70	**93.50**	80.30	98.50	**93.10**	**91.70**	52.50	**83.00**	**88.70**
	MV 10.WMV	**80.80**	**80.80**	93.40	**80.70**	**98.60**	93.00	91.50	**52.70**	82.90	88.60
Medium	Bagging 10.NMV	74.40	72.90	**93.50**	77.70	97.80	91.50	84.00	46.50	76.50	86.80
	Bagging 10.WMV	75.60	74.00	93.40	78.00	97.80	92.00	84.10	47.30	76.60	87.10
	Bagging 100.NMV	78.90	74.50	93.20	79.00	97.70	91.50	85.20	51.80	78.50	88.40
	Bagging 100.WMV	79.10	74.50	93.20	79.10	97.70	91.40	85.50	52.10	78.50	88.40

Table 4: Accuracies (%) of bagging and majority voting ensembles and best baseline models, and baseline model means. Ensemble size (10 or 100) and majority voting type (NMV or WMV) are marked after the ensemble type (Majority voting (MV) or Bagging).

6 Discussion and Conclusions

As is evident in Table 4, the best accuracies were obtained mostly by naive and weighted majority voting ensembles. In general, the results of the bagging experiments were slightly worse than both naive or weighted majority voting ensembles', but they were still mostly better than the results of individual baseline models. Whereas on high training data setting the bagging accuracies were slightly worse or the same when comparing to the naive and weighted majority voting ensembles, the differences were bigger on medium training data setting. For example, the difference between the best bagging ensemble and the best majority voting ensemble is greater than 2%-points for three languages (Khaling, Navajo, Russia). In general, bagging ensembles consisting of 100 models did deliver improvements upon ensembles consisting of 10 models. The results contain only one case where the bagging accuracy beats the accuracy of a majority voting ensemble (German, high training data), and even then the difference is only 0.1% which could just be due to random fluctuation. For one language (Khaling, high training data setting),

the overall best accuracy was obtained by a baseline model.

It is not entirely clear why bagging doesn't work well. One possible reason could be that the encoder-decoder models are so robust that they overfit too much with the sampled training sets that have only portion of the original examples but multiple duplicates. It is possible that having duplicates in each bagging resample could boost the overfitting problem. That is, each model would be a weak learner in its own way, and an ensemble of "bad" learners can't perform well. Another reason could be that there is simply not enough models in the bagging ensembles. As can be seen in Table 4, the performance of bagging ensembles of 100 models is often clearly better than ensembles of 10 models. Of course, training models for ensembles consisting of, for example, 1,000 or 10,000 models, would require a lot of computational resources and would start to get highly impractical. In addition, a regularization technique called dropout was used during the training of each model. Dropout ignores or "drops out" some number of layer outputs and aims to reduce variance similarly to bagging. It could be possible that gains from bagging are outshadowed by the use of dropout.

For future works, in addition to choosing only unique examples for bagging resamples and a larger number of models in ensembles, a more exhaustive search over the hyperparameter space could increase the performance of individual models and thereby result in more powerful ensembles. Hyperparameter tuning was not a priority in this study and only a limited amount of time was available. Furthermore, there are other ensemble methods that could be experimented with such MI RNNs. Even though the bagging results weren't as good as expected, using other ensemble methods, like boosting, could be worth exploring.

In summary, this thesis compares the performance of two different types of ensembles, bagging ensembles and majority voting ensembles, consisting of morphological inflection neural network models. Bagging didn't work as well as expected since majority voting ensembles performed better with nearly every language and training data setting. Thus, bagging – or at least the variant used in the experiments – is not the best option for achieving the best accuracies with morphological inflection RNNs. However, the results demonstrate that an ensemble of models trained in parallel nearly always outperforms a single model.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, abs/1409.0473, 2014.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs. *CoRR*, abs/1508.00657, 2015.

Leo Breiman. Bagging Predictors. *Mach. Learn.*, 24(2):123–140, August 1996. ISSN 0885-6125. doi: 10.1023/A:1018054314350.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR*, abs/1409.1259, 2014.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural Language Processing (almost) from Scratch. *CoRR*, abs/1103.0398, 2011.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, David Yarowsky, Jason Eisner, and Mans Hulden. The SIGMORPHON 2016 Shared Task—Morphological Reinflection. In *Proceedings of the 2016 Meeting of SIGMORPHON*, Berlin, Germany, August 2016. Association for Computational Linguistics.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Gėraldine Walther, Ekaterina Vylomova, Patrick Xia, Manaal Faruqui, Sandra Kübler, David Yarowsky, Jason Eisner, and Mans Hulden. CoNLL-SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection in 52 Languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 1–30. Association for Computational Linguistics, 2017. doi: 10.18653/v1/K17-2001.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Arya D. McCarthy, Katharina Kann, Sebastian J. Mielke, Garrett Nicolai, Miikka Silfverberg, David Yarowsky, Jason Eisner, and Mans Hulden. The CoNLL-SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection. *CoRR*, abs/1810.07125, 2018.

Mathias Creutz and Krista Lagus. Unsupervised Models for Morpheme Segmentation and Morphology Learning. *ACM Trans. Speech Lang. Process.*,

4(1):3:1–3:34, February 2007. ISSN 1550-4875. doi: 10.1145/1187415.1187418.

Michael Denkowski and Graham Neubig. Stronger Baselines for Trustable Results in Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 18–27, Vancouver, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3203.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993.

Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14 (2):179–211, 1990.

Pablo M. Granitto, Pablo F. Verdes, and H. Alejandro Ceccatto. Neural network ensembles: Evaluation of aggregation algorithms. *CoRR*, abs/cs/0502006, 2005.

L. K. Hansen and P. Salamon. Neural Network Ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10):993–1001, October 1990. ISSN 0162-8828. doi: 10.1109/34.58871.

Daniel Hernández-Lobato, Gonzalo Martínez-Muñoz, and Alberto Suárez. How large should ensembles of classifiers be? *Pattern Recognition*, 46(5): 1323 – 1336, 2013. ISSN 0031-3203.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Katharina Kann and Hinrich 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*, pages 62–70. Association for Computational Linguistics, 2016. doi: 10.18653/v1/W16-2010.

Katharina Kann, Ryan Cotterell, and Hinrich Schütze. Neural Multi-Source Morphological Reinflection. *CoRR*, abs/1612.06027, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

Christo Kirov and Ryan Cotterell. Recurrent Neural Networks in Linguistic Theory: Revisiting Pinker and Prince (1988) and the Past Tense Debate. *CoRR*, abs/1807.04783, 2018.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. ACL*, 2017. doi: 10.18653/v1/P17-4012.

Daniel Kondratyuk, Tomas Gavenciak, Milan Straka, and Jan Hajic. LemmaTag: Jointly Tagging and Lemmatizing for Morphologically-Rich Languages with BRNNs. *CoRR*, abs/1808.03703, 2018.

Kimmo Koskenniemi. A General Computational Model for Word-form Recognition and Production. In *Proceedings of the 10th International Conference on Computational Linguistics*, COLING '84, pages 178–181, Stroudsburg, PA, USA, 1984. Association for Computational Linguistics. doi: 10.3115/980431.980529.

Richard Maclin and David W. Opitz. Popular Ensemble Methods: An Empirical Study. *CoRR*, abs/1106.0257, 1999.

Alireza Mahmoudi, Mohsen Arabsorkhi, and Heshaam Faili. Supervised Morphology Generation Using Parallel Corpus. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 408–414, Hissar, Bulgaria, 2013. INCOMA Ltd. Shoumen, BULGARIA.

Peter Makarov and Simon Clematide. UZH at CoNLL-SIGMORPHON 2018 Shared Task on Universal Morphological Reinflection. In *CoNLL Shared Task*, 2018.

Peter Makarov, Tatiana Ruzsics, and Simon Clematide. Align and Copy: UZH at SIGMORPHON 2017 Shared Task for Morphological Reinflection. *CoRR*, abs/1707.01355, 2017.

Sarah Moeller, Ghazaleh Kazeminejad, Andrew Cowell, and Mans Hulden. A Neural Morphological Analyzer for Arapaho Verbs Learned from a Finite State Transducer. In *Proceedings of the Workshop on Computational Modeling of Polysynthetic Languages*, pages 12–20. Association for Computational Linguistics, 2018.

R. P. Neco and M. L. Forcada. Asynchronous translations with recurrent neural nets. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 4, pages 2535–2540 vol.4, 1997. doi: 10.1109/ICNN.1997.614693.

Steven Pinker and Alan Prince. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition*, 28(1):73 – 193, 1988. ISSN 0010-0277. doi: https://doi.org/10.1016/0010-0277(88)90032-7.

D. E. Rumelhart and J. L. McClelland. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2. chapter On Learning the Past Tenses of English Verbs, pages 216–271. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-13218-4.

Robert E. Schapire. The Strength of Weak Learnability. *Mach. Learn.*, 5(2): 197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. *CoRR*, abs/1409.3215, 2014.

Géraldine Walther, Guillaume Jacques, and Benoît Sagot. Uncovering the inner architecture of Khaling verbal morphology. 09 2013.

Shuly Wintner. Strengths and Weaknesses of Finite-state Technology: A Case Study in Morphological Grammar Development. *Nat. Lang. Eng.*, 14(4):457–469, October 2008. ISSN 1351-3249. doi: 10.1017/S1351324907004676.

Nasser Zalmout and Nizar Habash. Don't Throw Those Morphological Analyzers Away Just Yet: Neural Morphological Disambiguation for Arabic. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 704–713. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1073.