UTRECHT UNIVERSITY

BACHELOR THESIS

Enriching training data with syntactic knowledge and the effect on performance of a neural network on natural language processing tasks

---

Author:
Yuri Teerlink

Supervisor:
dr. Gijs Wijnholds

Universiteit Utrecht

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Faculty of Humanities
Utrecht University

January 23, 2021

UTRECHT UNIVERSITY

Abstract

Faculty of Humanities
Utrecht University

Bachelor of Science

Enriching training data with syntactic knowledge and the effect on performance of a neural network on natural language processing tasks

by Yuri Teerlink

Compared to neural networks (NN), humans can learn new concepts using only very little data. The ability to learn so efficiently might be due to the use of abstractions. To find similarities between human and machine learning this research will analyze if NN benefits from syntactic information during training. We will aim to answer the following question: How does enriching training data with syntactic knowledge affect the performance of a NN on natural language processing tasks? This research examines the results of Long Short Term Memory models (LSTM) trained on two different types of datasets; one without Part of Speech tags (a form of abstract knowledge) and a dataset that is supplemented with POS-tags. The results show that an LSTM trained on a relatively small dataset supplemented with POS-tags outperforms an LSTM trained on a regular dataset. The increase in performance might suggest that neural networks benefit from abstract information, which in turn might show some similarities in the way humans and machines learn.

Acknowledgements

First, I would like to express my appreciation towards my supervisor Gijs Wijnholds, for helping me all the way through and continually finding new ways to motivate and help me along with this project. Vicky, for carefully reading and providing valuable feedback. Ruben van Doorn, for giving the final push in the right direction. Wout and Koen, for their continuous help and motivation.

Contents

Chapter 1

Introduction

Human beings are capable of learning new concepts using very little data. For example, a child learning the difference between two objects will be able to differentiate the two after looking at only a couple of examples (Bloom, 2000). If we compare this to a neural network attempting the same task it will need a lot more training examples than the child. One explanation for the fact that humans can learn so efficiently, is the use of abstractions. Our ability to make abstractions allows us to generalize information better and therefore learn faster (Tenenbaum et. al., 2015). Besides being able to distinguish objects based on very little data, humans, especially children, are capable of learning new languages extraordinarily fast (Carey, 2009). This impressive performance on language can be mainly attributed to the use of abstractions, in this case, the use of syntactic knowledge (Baroni, 2019).

Gaining an understanding of the way humans learn is a broad topic of research within the field of artificial intelligence. Researchers hope to gain more understanding of how humans learn by creating models, such as neural networks, which learn (Lake et. al., 2017). If for instance, we see that a model performs better using some form of abstract knowledge, this might suggest it is learning similar to humans. A field of research where such models are being studied is Natural Language Processing (NLP). Here, language models are being trained to execute various NLP-tasks. These language models perform exceptionally well on various NLP-tasks: the machine-translation engine behind Google Translate [[1]] is improved each day (Läubli and Orrego Carmona, 2017), chatbots are created that can mimic humans almost seamlessly (Adiwardana et al., 2020) and OpenAI's GPT-3 language model is capable of human-like creative writing (Branwen, 2020).

Where humans do not require much input, previous and current NLP requires enormous datasets to learn and improve. This does not only increase training time, it is also not representative of human learning. As human learning is based on abstractions (syntactic information), a possible improvement for NLP is to supplement syntactic information to training sets.

Therefore, this thesis aims to supplement syntactic knowledge to training data and analyze whether this improves the performance of NLP. This aim results in the following research question:

How does enriching training data with syntactic knowledge affect the performance of a neural network on natural language processing tasks?

[1]https://translate.google.com/

1.1 Structure of this paper

We will briefly give an outline of how this paper is structured. Chapter 2 will provide the theoretical background knowledge relevant to this paper, in specific (recurrent) neural networks models, language models, abstractions, NLP-tasks, and the relation between human and machine learning. Chapter 3 will discuss the methods that will be used. Chapter 4 will present the results obtained during the research. And finally, we will conclude with a summary in Chapter 5.

Chapter 2

Theoretical Background

This chapter provides a brief description of the most important technical and linguistic concepts that will be discussed in this thesis. General technical concepts regarding neural networks are not included in this chapter assuming the reader is aware of these. However, recurrent neural networks and long short term memory models will be discussed in detail as these lay the foundation for the experiment which will be performed. The linguistic concepts that will be discussed are part-of-speech tags, these tags will be used as syntactic information in this experiment, and parse trees, which have been the subject of previous research. Finally, the chapter will conclude with two different types of natural language processing tasks that can be used to measure the performance of a neural network. The first NLP-task that will be discussed is named entity recognition, which is used in this experiment. The second task is part-of-speech tagging, a widely used NLP-tasks on which neural networks reach near-human performance.

2.1 Neural networks and their success

Ever since the thaw of the so-called 'AI Winter' (Howe, 2007), there has been an increase in the usage and research of neural networks. The sudden increase in popularity can be attributed to the increase in computing power and increasingly larger datasets being available (Smith et al., 2006). However, regular feedforward neural networks do have some drawbacks, as they tend to underfit and underutilize computing resources (Lipton and Berkowitz, 2015). Nevertheless, there is no denying the apparent success within the field of AI. Relevant examples are gameplaying systems such as AlphaZero and AlphaGo (Silver et al., 2017), IBM's Watson (Ferrucci et al., 2010), and Google Translate's [1] performance on language translation. What caused the success within the field of AI? In order to better utilize the available computing power and larger datasets, more complex models had to be designed, such as different kinds of deep learning architectures (i.e. convolutional neural networks, recurrent neural networks).

Another limitation of regular feedforward networks is that there is no ideal way for a neural network (NN) to process sequential data. Take for example an image classification task: a neural network would be perfectly capable of identifying whether a certain object or person appears in one given image. However, if we make

[1] https://translate.google.com/

things more complicated, for example, by asking a network to determine the movement of a person in a given sequence of images, it would have no way of producing output at each given timestep. The reason why this is hard for a regular NN, is due to the assumption of time independence, this makes it impossible for the network to maintain a time sequence in order.

2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) offer a solution to the incapacity of NN's to deal with temporal sequences. RNNs have a hidden state vector which will be updated each iteration. This section will go into greater detail about how the hidden layer operates.

Similar to feedforward networks, recurrent neural networks consist of a connected feedforward network, in which cycles are not allowed. However, what differentiates an RNN from a regular NN, is the fact that an RNN possesses an additional hidden layer consisting of recurrent nodes. This hidden state vector allows the network to "remember" a previously seen state and keep track of the previous state, as a means to more effectively process sequential data. In contrast with the non-cyclic property of feedforward neural networks, cycles are allowed within the hidden layer in an RNN. Because of the occurrence of cycles, a datapoint at timestep $t-1$ may influence a datapoint at timestep t. For instance, at timestep t, nodes within the hidden layer $h(t)$ not only get information from the current data point $x^{(t)}$, but also from the hidden layer $h^{(t-1)}$ of the network's previous state. Moreover, the output $y^{(t)}$ at each time t is determined by the current hidden node $h^{(t)}$. Since $h^{(t})$ is partly based on $h^{(t-1)}$, which in turn is based on $x^{(t-1)}$ it becomes clear how previously seen data can influence data being processed at time step t. The following equations are necessary for the calculations of the process in an RNN:

$$\mathbf{h}^{(t)} = \sigma(W^{hx} \cdot \mathbf{x}^{t} + W^{hh} \cdot \mathbf{h}^{(t-1)} + \mathbf{b}_{h}) \tag{2.1}$$

$$\hat{\mathbf{y}} = softmax(W^{yh} \cdot \mathbf{h}^{(t)} + \mathbf{b}_{y}) \tag{2.2}$$

Where W^{hx} is the conventional weight matrix for the input layer, W^{hh} is the matrix corresponding to the recurrent weights, itself, and the nodes of the following time steps. W^{yh} is the weight matrix corresponding to the given output layer at each time step. Lastly, $\mathbf{b}_{h}$ and $\mathbf{b}_{y}$ correspond to the bias weights.

The availability of $\hat{\mathbf{y}}$ at each timestep t allows an RNN to produce output at each timestep, allowing data to be processed as shown in Figure 2.1

The recurrence of RNN comes at a cost, however. The gradient is prone to problems, such as exploding or vanishing while backpropagating (Hochreiter et al., 2001). During backpropagation, the gradient is computed by taking the derivative. Consider a network with n layers, this will cause n derivatives to be multiplied together. In the case where the derivative is large, multiplying will only increase the

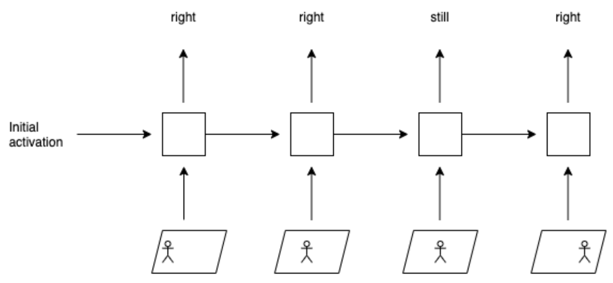

FIGURE 2.1: A systematic overview of an RNN processing sequential data one image at a time, where the ingoing arrows represent the input at each timestep and the outgoing arrows represent the corresponding label that is predicted.

gradient to the extent that the value will keep on increasing exponentially, causing a so-called 'exploding gradient'. In contrast, if the derivative becomes sufficiently small, multiplying it will cause the gradient to converge to zero, corresponding to the vanishing gradient problem.

2.3 Long Short Term Memory Model

In 1997, Hochreiter and Schmidhuber first introduced their Long Short Term Memory (LSTM) model as a way to handle the vanishing gradient problem. At first sight, an RNN and LSTM do not differ that much from each other, as both have a hidden layer. What makes LSTMs unique, however, is that each node in a hidden layer is what Hochreiter and Schmidhuber call a "memory cell" (Hochreiter and Schmidhuber, 1997). The self-recurrent edges within the memory cell allow the gradient to pass over the cell without vanishing or exploding.

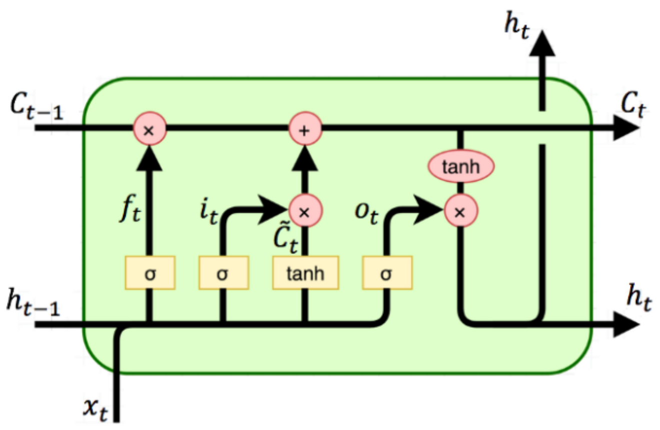

FIGURE 2.2: *A systematic overview of a memory cell:* The current hidden and cell state, h_t and c_t, are updated using the hidden and cell state of the previous iteration, h_{t-1}, and c_{t-1}. Image obtained from https://towardsdatascience.com/predictive-analysis-rnn-lstm-and-gru-to-predict-water-consumption-e6bb3c2b4b02

Figure 2.2 gives a systematic overview of the memory cells which replace the standard hidden nodes of an RNN. Furthermore, the following formulas show all the computations involved within the memory cell.

One iteration over the memory cell goes as follows: the previous hidden state h_{t-1} is combined with the current input vector x_t and passed along to the forget gate of the memory cell, as shown in Figure 2.2. Within the forget gate, x_t and h_{t-1} are multiplied with their respective weight matrices, the sum of this multiplication is passed through a σ-function. The resulting vector f_t is multiplied with the previous cell state c_{t-1}: their product represents what information is desired to remain and what can be forgotten. Next, the cell state must be updated, which takes two steps. Firstly, x_t and h_{t-1} are passed through the input gate to determine which values will be updated. Secondly, a vector containing new candidate values is computed within the *tanh*-layer, giving us $\widetilde{C}_t$. Now the cell state can be updated by adding the product of the forget gate f_t and C_{t-1} to the product of the input gate i_t with $\widetilde{C}_t$. The output y_t of the current time step is equal to the current hidden cell state. h_t is computed by taking the product of the output gate o_t and the tanh of the current cell state C_t. The cell state C_t, together with the hidden state h_t, will be passed along to the next node in the network.

2.4 Syntactic Information

Abstractions are at the heart of this research. The type of abstractions that will be used are the ones that are common in grammar. Two common types of syntactic information are parts of speech and parse trees.

2.4.1 Part of Speech

Traditional grammar has a way of classifying words in different word categories depending on their respective syntactic and morphological properties. The collective name for these word categories is part of speech (POS). Classifying words in different word categories gives more information and insight about their use and properties. For example, verbs can express an action, nouns generally will denote things and objects, while adjectives give information about the qualities of these things and objects. The most commonly used parts of speech are noun, verb, adjective, pronoun, preposition, conjunction, interjection, numeral, and determiner. Besides these nine parts of speech, there is a wide variety of different levels of separation depending on how precise a denotation is required.

2.4.2 Parse Trees

Parse trees are directed tree graphs that represent the hierarchical structure of a sentence. Figure 2.3 gives an example of how a parse tree can represent a sentence. A node within a parse tree functions as a token for a word within a sentence, the edges

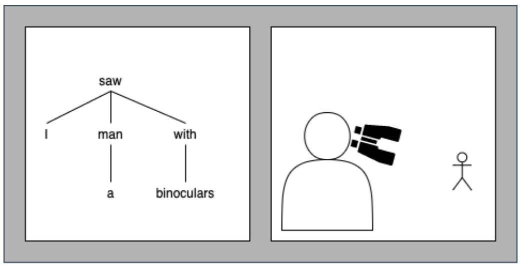

FIGURE 2.3: *Left:* a possible parse tree for the sentence "I saw a man with binoculars." *Right:* visual representation of the corresponding sentence

represent the syntactic relation between the tokens. Parse trees show the underlying structure of a sentence and the relation between its constituents.

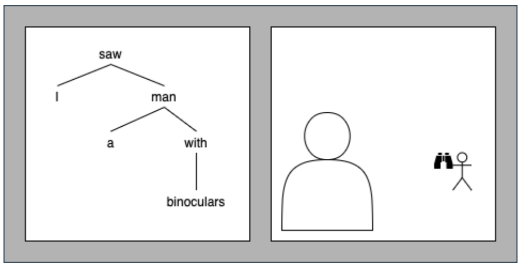

FIGURE 2.4: *Left:* the second possible parse tree for the sentence "I saw a man with binoculars." *Right:* visual representation of the corresponding sentence

However, the sentence “I saw the man with binoculars” is an example of what is known as an ambiguous sentence i.e., it has more than one interpretation. In this case, it is unclear as to who exactly is holding the binoculars. This is one useful application for the use of parse trees. Using the parse tree shown in Figure 2.3 it becomes clear that the “I” in the given sentence is the one holding the binoculars resolving ambiguity. Figure 2.4 gives another parsing of the same sentence, here, it is the man holding the binoculars as shown in the corresponding visualization. Depending on the context we can choose the corresponding parse tree to appropriately resolve the ambiguity and give the sentences their proper meaning.

2.5 Natural Language Processing Tasks

There are two natural language processing (NLP) tasks that are of interest to this research. NLP-tasks are used to measure the performance of a language model, in this case, an LSTM.

2.5.1 Named Entity Recognition

Named entity recognition is an NLP-task focused on information extraction of unstructured text. The goal of this task is to identify words in a given sentence that are a named entity i.e. a person, location, organization, monetary value, time expression, etc. Take for example the following sentence:

Vlaanderen onderschrijft expliciet de doelstellingen van de UNESCO .

Then the correct corresponding named entities will be:

(*Vlaanderen*, B-loc), (*onderschrijft*, O), (*expliciet*, O), (*de*, O), (*doelstellingen*, O), (*van*, O), (*de*, O), (*UNESCO*, B-org), (., O)

As shown in the example above the named entities are labeled accordingly. The labels consist of two parts separated with an "-". The first character indicates whether the labeled word is the first word of a named entity as is the case in the given example. In the case of a named entity that consists of multiple words the first label of the first word will contain a "B" and the other labels will start with an "I". The second part of the label denotes the type of entity the word belongs to, again looking at the example above "loc" in the label "Vlaanderen" corresponds to location, and "org" in the label of "UNESCO" stands for organization.

2.5.2 Part of Speech Tagging

Like named entity recognition, part of speech tagging is a task where for a given sentence each word needs to be correctly classified with its corresponding label. As the name suggests, with part of speech tagging a trained language model is tasked to correctly predict the part of speech of each word within a sentence. Classifying the same sentence that was shown for the NER tasks but now with part of speech tags gives us:

(*Vlaanderen*, N), (*onderschrijft*, WW), (*expliciet*, ADJ), (*de*, LID), (*doelstellingen*, N), (*van*, VZ), (*de*, LID), (*UNESCO*, N), (., LET)

The following chapter will go into greater details about how the experiment is performed, the data and model will be used and, how the experiment is evaluated.

Chapter 3

Methods

This experiment tries to show whether there is a difference in performance between a network that has had access to syntactic information during training and one that was purely trained on the words. This will lead to four networks being trained for the NER task. The first network will be trained on a large dataset consisting only of sentences with (word, label) tuples. The second network will be trained on a smaller dataset consisting of sentences but with an added part of speech data. The third network will again only use sentences with the word, label information but this time using the smaller dataset. Lastly, a network will be trained on sentences where the words will be ignored and the network only has access to part of speech and label data. The code that is used for this experiment is available at [[1]].

3.1 SoNaR-500 Dataset

The dataset that will be used in this experiment is the SoNaR-1 dataset which is a subset of the Dutch SoNaR-corpus SoNaR-500, which is a data set that is fully lemmatized and annotated with part of speech tags. The SoNaR-1 data set consists of approximately one million words and is provided with various kinds of semantic annotations (e.g. named entities, spatial and geotemporal relations). For more information about the SoNaR-corpus see [[2]].

For this experiment different partitions of the named entity and part of speech task were chosen. The first network will be trained on the full named entity dataset which amounts to 39,149 sentences for the training, 8,649 for the validation, and 8,538 for the test set. For the following parts of the experiment, we combined the sentences from the named entity with those that also occur in the part of speech dataset. The intersection of both datasets amounts to 5,221 sentences for the training, 1,120 for the validation, and 1,118 for the test set. Table 3.1 shows a distribution of the named entity labels for both datasets and Table 3.2 shows the labels and their meaning.

[1]https://github.com/iehkaatee/LSTM_project_ner
[2]https://taalmaterialen.ivdnt.org/download/tstc-sonar-corpus/

TABLE 3.1: Number of labels for each category in the full and combined dataset.

NER label	Full dataset	Combined dataset
B-loc	25461	2901
B-pro	3768	337
I-pro	5686	837
B-per	14077	1896
B-org	9750	1629
B-misc	7114	1075
I-misc	2377	366
I-org	7014	1199
I-per	9111	1276
I-loc	2562	256
B-eve	1077	61
I-eve	1041	52

TABLE 3.2: NER labels and their corresponding description.

NER label	Description
"B" prefix	Begin named entity
"I" prefix	Continuation named entity
loc	Location
per	Person
pro	Product
org	Organization
misc	Miscellaneaous
eve	Event
O	Other

3.2 Long Short Term Memeory Model

The LSTM model that is used for this experiment is implemented in python using the PyTorch library[3]. PyTorch provides many ready-to-use models that can be easily implemented and modifiable to a wide variety of projects. This experiment will require two different kinds of LSTM models, one for processing the data without POS-tags and one for data with POS-tags. The model that will only train on (word, label) data as input is a straightforward basic LSTM model without any special adjustments, see Figure 3.1 for a schematic overview of the model. However, feeding the combined data to the network required an additional embedding layer dedicated to the part of speech tags. Figure 3.2 shows an adjusted LSTM and how the embedding layer of the parts of speech is combined with that of the word embedding and is passed to the memory cell of the LSTM. The combined embedding layer is the result of concatenating the word embedding layer with the part of speech embedding layer to create one combined embedding layer. The Adam optimizer (Kingma and Ba, 2014) was used with a learning rate of a=0.001.

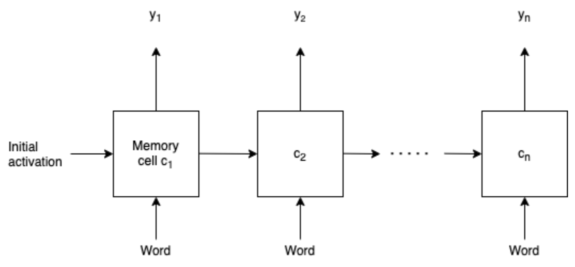

FIGURE 3.1: A systematic overview an standard LSTM processing only the given words at each timestep.

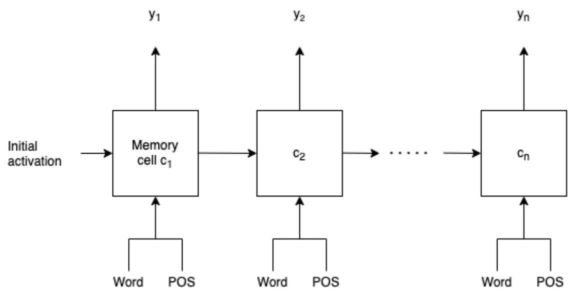

FIGURE 3.2: A systematic overview of the altered LSTM processing the combined word and POS data.

3.3 Hyperparameters

The hyperparameters were chosen using a parameter sweep trying different values step by step, keeping track of which settings improved performance. For the number

[3]https://pytorch.org/

of epochs, we started at 5 and kept increasing the number of epochs by 5 each time until there was no more significant increase in performance. The same was done for batch sizes here we started at 100 and kept increasing by 100 each run. The embedding and hidden dimension layers started from 20 and were increased by 10 each run. Due to limitations with computing power, the parameters are only optimized for one model. The optimal parameters found for this model will be used for the other models as well. This resulted in the following parameters being used:

- Epochs: 40
- Batch size: 200
- Hidden dimension: 82
- Embedding dimension: 82
- Learning rate: $\alpha = 0.0001$

3.4 Evaluation

One important thing to take into consideration during the evaluation was the fact that most words within a sentence are not a named entity (label = "O"). Therefore, when evaluating the performance of the models including "O"-labels will always perform reasonably well due to the frequency of the "O" labels. To correctly measure the performance of the correct named entities that were predicted the choice was made to neglect the "O" labels (for other) and only look at the labels the network guessed correctly. The metrics that are used to measure performance are F1-score, precision, and recall.

Chapter 4

Results

The Combined dataset without POS (M_1) showed baseline precision values of 0.493 (precision), 0.499 (recall) and 0.496 (F1). As a control, a model was trained (M_2) on a full dataset showing improvement of all parameters as was expected from a larger dataset.

After supplementation of POS (M_3) precision, recall and F1 were increased by approximately 10% which is almost equal to the results of M2, trained on the full dataset. Lastly, a model trained on only pos performed nowhere close to the other models.

Model		Precision	Recall	F1-score
M_1:	combined dataset, without POS	0.493	0.499	0.496
M_2:	full dataset, without POS	0.562	0.549	0.553
M_3:	combined dataset, with POS	0.549	0.565	0.557
M_4:	combined dataset, only POS	0.661	0.234	0.346

TABLE 4.1: Precision, recall and F1-score of each of the models achieved on NER.

Chapter 5

Discussion

This research investigated whether an LSTM trained on data that is supplemented with syntactic information will increase its performance. To test this, we performed several experiments and compared the results of different training compositions. First, we compared models trained on the combined or full dataset showing an increase of all parameters. This was an expected finding, as the full dataset consisted of four times the training examples.

The major result of these experiments is that an LSTM trained on the enriched dataset performs significantly better than a model that is trained on a regular dataset of the same size (M_1: $f1 = 0.557$ vs. M_2: $f1 = 0.496$). A possible explanation for the increase in performance is that syntactic information, in this case, part-of-speech tags, helps the network differentiate between what is a named entity and what is not. Intuitively this makes sense, for example, knowing whether a given word is a verb makes it less likely that that word will be a named entity. The increase in performance might suggest that syntactic information positively affects a neural network's ability to differentiate named entities from non-named entities. Another interesting result was that even though the enriched dataset was significantly smaller in size (size dataset) this did not cause the network that was trained on the smaller set to underperform in comparison with the network trained on the larger dataset. On the contrary, the network that was trained on a smaller dataset supplemented with syntactic information even has a slightly higher performance than the network trained on the larger dataset without syntactic information (M_1: $f1 = 0.557$ vs. M_3: $f1 = 0.553$). This second observation indicates that supplementing data with syntactic information might increase the rate at which neural networks train. Furthermore, in line with the expectations, a model trained on only POS data performed subpar compared to the other models. Intuitively this makes sense because by using only POS-tags it will be harder to distinguish words that share the same POS-tag.

Limitations

Several limitations should be noted. The first is that the performance of the networks trained for this experiment is nowhere near the performance of some of the state-of-the-art models that exist today. For instance, Chiu and Nichols's bidirectional LSTM with character-level features obtains f1-scores upwards of 90% (Chiu and Nichols, 2016). The difference in the performance of the network that is trained in this study and that of Chiu and Nichols networks can be explained by multiple reasons. The first is the size of the dataset, the initial dataset contained 56,336 sentences and the combined dataset even contained as few as 7,459 sentences, if we compare this to the dataset used by Chiu and Nichols: CoNLL-2003 (train: 204,567, dev: 51,578, test: 46,666) and OntoNotes 5.0 (train: 1,088,503, dev: 147,724, test: 152,728) this difference becomes apparent. That a network that is trained on a significantly larger dataset performs better should come as no surprise. Furthermore, due to limitations in computing power, this experiment used a network with a simple architecture, examples show that complex architectures can perform significantly better.

Chapter 6

Conclusion

This research analyzed whether an LSTM model trained on a dataset that is supplemented with syntactic information increases its performance on named entity recognition. Our results show that not only does a model that is trained on an enriched dataset outperform a model trained on a regular dataset, the model that was provided with syntactic information needed only a fraction of the number of training examples to achieve the same results as a model trained without. Still, there is still much room for improvement for further research. For example, this experiment made use of a relatively basic neural network architecture, an LSTM model that is losing in popularity to newer models like transformers (Vaswani et al., 2017) which show promising performances in the field of natural language processing. Furthermore, since combining the part-of-speech dataset with the regular dataset resulted in a relatively small dataset it would be worth studying the performance of a similar experiment but using a larger dataset. Nevertheless, the increase in performance of a network trained with the help of syntactic knowledge might suggest that a neural network can benefit from the use of abstractions. If it is the case that neural networks, like humans, make use of abstractions, this suggests that there might be some similarities in the way neural networks and humans learn. Taken together, supplementing a dataset with a form of abstract information seems well worth the effort.

References

Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., ... others (2020). Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*.

Baroni, M. (2020). Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society B*, *375*(1791), 20190307.

Bloom, P. (2002). *How children learn the meanings of words*. MIT press.

Branwen, G. (2020). Gpt-3 creative fiction.

Carey, S. (2000). The origin of concepts. *Journal of Cognition and Development*, *1*(1), 37–41.

Chiu, J. P., & Nichols, E. (2016). Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, *4*, 357–370.

Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A. A., ... others (2010). Building watson: An overview of the deepqa project. *AI magazine*, *31*(3), 59–79.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.* A field guide to dynamical recurrent neural networks. IEEE Press.

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

Howe, J. (2007). Artificial intelligence at edinburgh university: A perspective. *Archived from the original on*, *17*.

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

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, *40*.

Läubli, S., & Orrego-Carmona, D. (2017). When google translate is better than some human colleagues, those people are no longer colleagues.

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... others (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Smith, C., McGuire, B., Huang, T., & Yang, G. (2006). The history of artificial intelligence.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *science*, *331*(6022), 1279–1285.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.