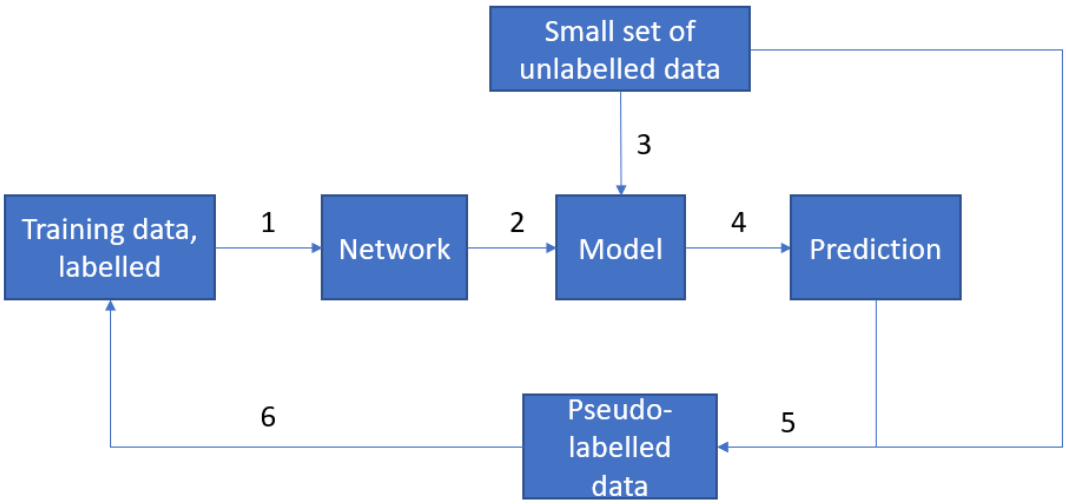
CHALMERS
UNIVERSITY OF TECHNOLOGY



Exploring semi-supervised learning for deciding the order of vehicles in rear-end crashes

Master's thesis in Algorithms, languages and logic

Linus Johansson

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:95

Exploring semi-supervised learning for deciding the order of vehicles in rear-end crashes

LINUS JOHANSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
Gothenburg, Sweden 2019

Exploring semi-supervised learning for deciding the order of vehicles in rear-end crashes
LINUS JOHANSSON

Supervisor: Selpi, Department of Mechanics and Maritime Sciences
Examiner: Selpi, Department of Mechanics and Maritime Sciences

Master's Thesis 2019:95
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of a semi-supervised architecture with self-training that is used in this project.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Exploring Semi-Supervised Learning for Deciding the Order of Vehicles in rear-end Crashes

LINUS JOHANSSON
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Knowing the order of vehicles in rear-end crashes is an important step for further analysis to work towards the goal of reducing fatalities and severe injuries in traffic down to zero. For example, by knowing the order of vehicles a more accurate estimation of the driver's exposure could be done. The main task of this project was to create a system which could decide the order of vehicles involved in each rear-end crash from the text description in a crash database. To try and address this task semi-supervised learning was chosen as the method. The reason for this choice was because all the crash cases were unlabelled from the beginning and to label all the cases manually would be infeasible but to label a few hundred cases was manageable.

Two neural networks were built and tested. The text descriptions of the rear-end crashes from the national crash database in Sweden were transformed to vectors and used as input for the neural networks. The performance of each of the networks was measured. The results suggest that further work are needed before the networks could be used for making reliable decisions, however the experiments give some indications on what parameters should be considered carefully while using semi-supervised learning and which parameters that need more exploration.

Keywords: Neural Networks, Semi-Supervised Learning, LSTM, CNN, Traffic Safety, Road Accidents database, Natural Language Processing, Deep Learning

Acknowledgements

I would like to thank my supervisor Selpi for all the support, motivation, ideas and recommendations she has given me throughout the project.
I would also like to thank Ron Schindler and Andras Bálint, researchers at the Vehicle and Traffic Safety Centre at Chalmers (SAFER), for helping me in the beginning with understanding the data from STRADA.

Linus Johansson, Gothenburg, July 2019

Contents

List of Figures

1
Introduction

Vision zero is a goal to have a traffic system with no fatalities or severe injuries. To work towards this goal researchers could use a method called *quasi-induced exposure* method. The idea of this method is to get a more accurate estimate of the driver's exposure [5]. However, the method needs identification of *not-at-fault drivers*, which is not readily available in the crash database but may be derived from the text description in the crash database. Therefore, there is a need of extracting not-at-fault drivers from the text data. The first main challenge is that everything is in text. Although computers are great with numbers they are not too good with text. One issue with language is that there could be ambiguous sentences where it is troublesome to derive the correct meaning of the sentence without looking at the context. Another problem could be irony or sarcasm, where the context also plays a major role. Grammatical information is also difficult to handle, for example it makes a difference if "vehicle A crashed into vehicle B" or "vehicle B crashed into vehicle A".

1.1 Goal and objectives

The main goal of this project was to implement a deep neural network that determines the order of vehicles involved in rear-end crashes from the text description of the Swedish national accident database (STRADA). If the order of vehicles involved are decided it could be assumed that the driver of the vehicle which got hit in the back is a not-at-fault driver. The reason for choosing neural network is that it has made many impressive results in many areas, such as natural language processing [20], playing games [12], image recognition [17] among others.
The first objective of this project was to transform the text data into readable data for the computer. The second objective was to build a reasonable neural network that can be used to achieve the main goal of the project. The third objective was to tweak the parameters then analyze and evaluate the models produced.
The challenge of this project is to interpret natural language in such a way that an order of items could be extracted. This is challenging due to the fact natural language could be ambiguous, but also because explicit information could be left out and a human would understand it by its context or by the order of things. As an example "vehicle A got flashed by the sun. She then collided with vehicle B" where it is easy for a human to see that *She* refers back to *vehicle A*, this might be very difficult for a learning system if there are not enough training examples present representing such cases.

1.2 Project Scope

The project will only focus on rear-end crashes and no other types of crashes. It also only focuses on rear-end crashes where only two vehicles are involved (i.e. crashes with more than two vehicles are not considered here).

1.3 Report Outline

The rest of this paper is outlined as follow. Chapter 2 explains how the data looks like, how the database STRADA is structured and also how the data is extracted. Chapter 3 covers the theory behind the technology that is used in this project. It briefly explains the main technique called semi-supervised learning and also neural networks. Chapter 4 describes what kind of methods are used and how the experiments are conducted. Chapter 5 reports the results from the experiments and Chapter 6 discusses the evaluations and comments about the result. Chapter 7 explains what kind of conclusions that could be drawn from the project and the possibilities for future work.

1.4 Related work

Results of literature search suggests that there has been no similar work trying to get the same information from other or similar databases. However there exists work which used similar methods used in this project but for different problems. One of these is medical text classification which utilizes a convolutional neural network (Section 3.3.2) to classify text [10]. Another one is weather forecasting which used a combination of long-short term memory neural network (Section 3.3.1) and a convolutional network [9]. A third one which also used a long-short term memory neural network called abstractive text summarization, which merged key sentences and made a summary out of them [3]. More of the related work can be read in Section 3.

2
Data

This chapter will present how the data looks like, how the data was extracted and how labelling of the data was done. The crash descriptions in STRADA are written in Swedish. However, for the purpose of illustration, the examples will be translated and shown here in English.

2.1 The data

The accident descriptions in STRADA differ highly in length (e.g., some have less than 10 words and some others may have more than 100 words). There is also no convention of how to write the descriptions. Some examples of how a sentence could look like is: ("passenger car 2 is hit from behind by the driver of passenger car 1.") ("Passenger car driven into passenger car in front when this car stopped for red lights in a crossing"). Some of the accident descriptions have indices to identify the vehicles involved in the crashes (like in the first example), but some others do not have (like in the second example). The STRADA data used here covers the period from 2003 to August 2017. In total there were 280354 police reports and 42722 of them were classified as rear-end crashes. Out of the 42722 there were 30548 crashes where there were only two vehicles involved. The rest had more than two vehicles. After filtering on the indices there were 26240 different crashes left. The content of the dataset can be seen in Figure 2.1 ; the subset that is circled in red is the subset that is used for further processing and analysis.

Three tables from STRADA were used, *Report*, *TrafficElements* and *Policereport* table which all can be seen in Figure 2.2. All three reports are connected with two ids, *ExtendedReport_id* and *report_id* ; these ids together represent a unique crash in the tables.

In the *Report* table all the text descriptions are written. The *PoliceReport* table contains information on the type of crash (e.g., rear-end crash, animal related crash, frontal crash, etc.) The *TrafficElements* table contains one row for each vehicle where each vehicle involved in a crash gets a unique *elem_id*. The *PrimaryElemtype* in the *TrafficElement* table is what kind of "vehicle" that *elem_id* was and the *Subelem_type* is a more specific description of the *PrimaryElemtype*. *PrimaryElemtype* could be motor vehicle and *Subelem_type* could be passenger car, truck, motorcycle and others. *PrimaryElemtype* could also be pedestrian, animal, trailer and some more. The *Primary_elem* in the *TrafficElements* table was a bit unclear what it was and there were no clear description of it.

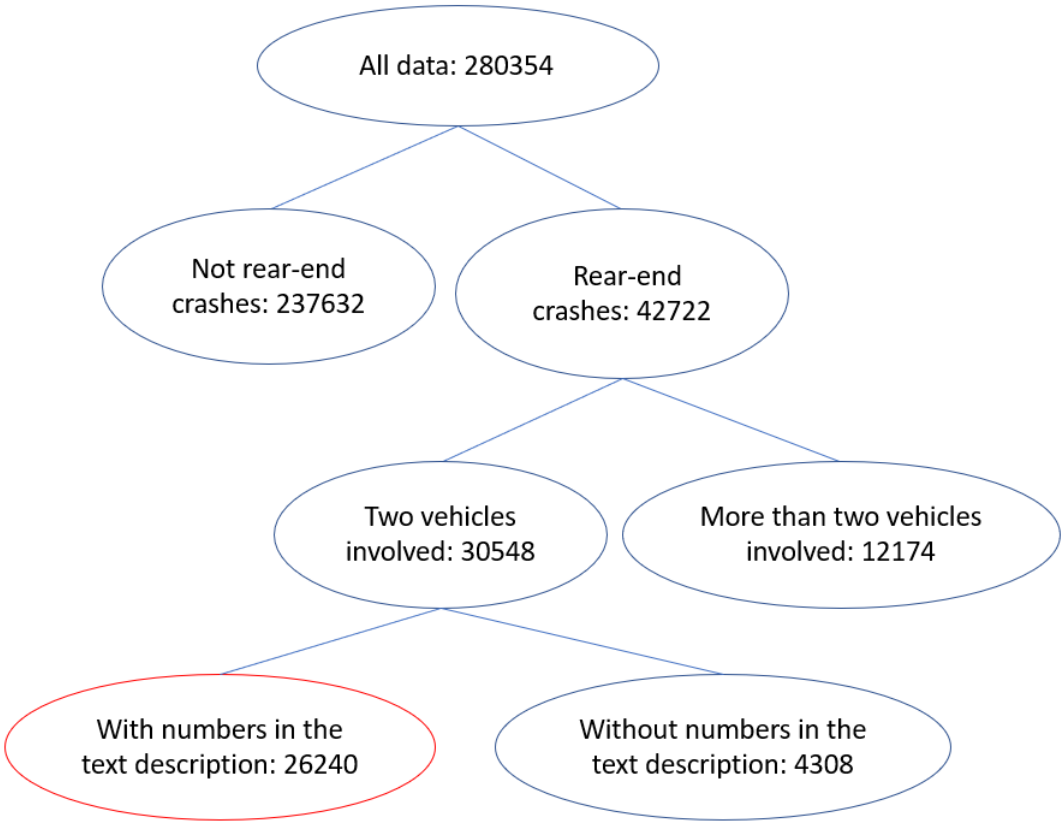


Figure 2.1: An overview of the content of the STRADA data from 2003 to August 2017.

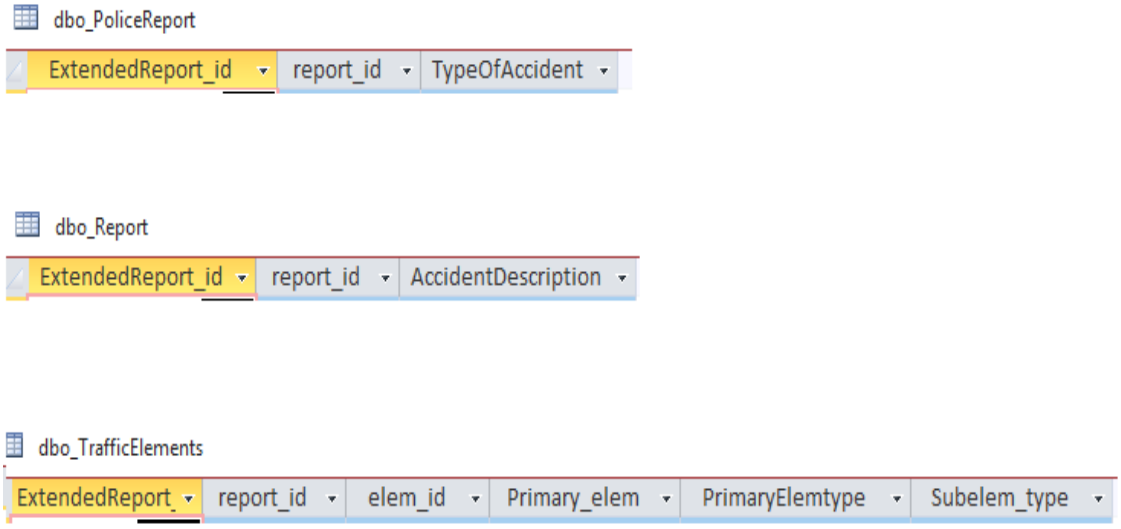


Figure 2.2: Overview of the tables used.

A small investigation were made by looking at both sketches(these sketches were not available for the project, but the knowledge were fetched from others) and the

text with indices. A comparison were made to see how well the indices in the text matched the indices in the sketches. Due to how time consuming this would be to look through all only 10 cases were looked through to see if there was any pattern. After this investigation it was assumed that the *elem_id* in the *TrafficElements* was the only connection between the index in the accident description and the tables. This assumption affected the way the extraction was done and which cases to keep.

2.1.1 Extraction

First, each crash was identified to be of the rear-end crash type and then the *ExtendedReport_id* and *report_id* were saved for these crashes. After that the ids were used to see how many times they occurred in the *TrafficElements* table to determine how many vehicles were involved. Lastly, to identify if there were indices in the text, each description was looked through automatically by a script to find a number in the text. If a number was found it was classified as a report with indices and if no number was found it was discarded. However, there exist cases where numbers were in the text but not used as indices, but were still classified as a report with indices. Examples of such cases are when there was road numbers and dates in the accident description.

2.1.2 Preparing labelled data

From the start there were no labelled data. The labels have been done manually to create some initial training data. The labels were done by reading a sample and asking the question, "is vehicle 1 in the front?", if yes, then 1 was assigned as a label to this sample, if no, 0 was assigned. First, 100 training samples and 20 test data were labelled. Later the test data set was extended to 60, then merged with the training data to get a bigger training set and a new test data set of 347 samples were labelled. In the end also these 347 test data were merged with the training data to give a training data set of 507 samples and a new 86 samples were labelled for the test data. In the end there were 507 training samples and 86 test samples.

3

Theory

This chapter will explain semi-supervised learning and deep neural networks.

3.1 Semi-supervised learning

In the context of machine learning, semi-supervised learning is a technique which is some what in-between supervised learning and unsupervised learning. Its main idea is to circumvent the need to have tons of labelled data and only needs a small set of training data.

There are several methods (e.g. self-training and co-training [22]) for semi-supervised learning, the one chosen in this project is called self-training. The procedure for how it works is quite simple and can be seen in Figure 3.1 and in Equation 3.1 [22].

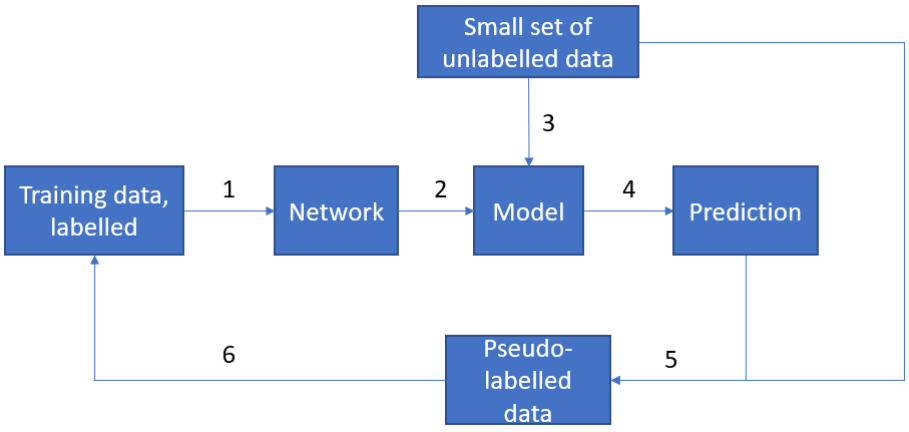


Figure 3.1: How a semi-supervised learning network works: 1. Training data is sent to the learner 2. Network creates a model 3. A small set of unlabelled data is sent to the model 4. The model makes predictions on the unlabelled data 5. The unlabelled data and the predictions creates a pseudo-labelled data set 6. The pseudo-labelled data is added to the training data

$$
\begin{aligned}
&\textit{Input: labelled data}(x_i, y_i)_{i=1}^{l}, \textit{unlabelled data}x_j{}_j^{l+u} = l+1 \\
&1.\textit{Initially, let}L = (x_i, y_i)_{i=1}^{l} \textit{and}U = x_j{}_j^{l+u} = l+1 \\
&2.\textit{Repeat:} \\
&3.\textit{Train } f \textit{ from } L \textit{ using supervised learning.} \\
&4.\textit{Apply } f \textit{ to the unlabelled instances in } U. \\
&5.\textit{Remove a subset } S \textit{ from } U\textit{; add}(x, f(x)|x \in S)\textit{to}L
\end{aligned}
\tag{3.1}
$$

How to divide the unlabelled and labelled data is not clear. An example from literature that has investigated the effect of the amount of unlabelled data and labelled data for a semi-supervised learning context is from Street View House Number (SVHN) project. SVHN is a dataset with 73257 labelled digits for training. In that project, comparison was made when training was done with 1%, 10%, 50% and 100% of the data as labelled and the rest as unlabelled. When the sets were highly unbalanced it was compensated by using every labelled example multiple times. The results produced with 1% as training data had 15.30% in error rate, 10% as training data got 7.94%, 50% got 5.65% and with 100% labelled data it was 4.60% [16].
Evaluating semi-supervised learning models is not trivial, but there are some recommendations and hints to make it easier [14].

- Small difference in the model or structure can impact the results greatly.
- To have a supervised setting to compare the performance with, since the goal is to outperform the supervised setting.
- That the semi-supervised learning network might suffer if the unlabelled data and the labelled data comes from different classes.
- The network benefits from seeing more data.

3.2 Neural networks

Neural networks is a method that tries to simulate how the brain works. Like when you touch the hot stove with your hand, your hand gets several signals e.g. temperature, which then are sent to the brain, the brain then makes a decision on what to do with the hand. Leave it there or remove it. Neural networks work in almost the exact same way, some input data, then the output should be some decision made based on the inputs.
Neural networks consists of nodes or neurons, and several layers of them. A node has several inputs and has one output as in Figure 3.2. Each input is multiplied with some weight, w, and a bias, b, is added before the input goes into the neuron. The neuron then uses some function, h, to produce the output y.

$$z_1 = x_1 w_1 + b_1$$

$$y_1 = h(z_1), \text{where h is some function}$$

By having multiple neurons stacked together we get layers and a neural network as in Figure 3.3.

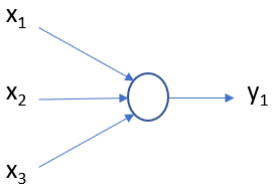


Figure 3.2: A single node, **x** as input and y as output

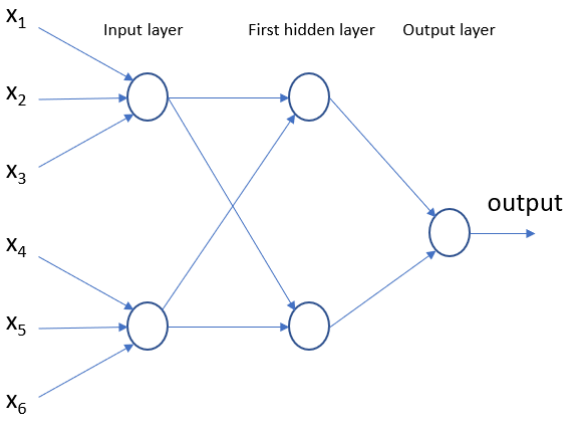


Figure 3.3: Simple neural network

A loss function which compares the true output against the network's output is also added in the end. This is necessary to be able to try and optimize the network by minimizing the loss. Minimization of the loss is usually done by gradient descent and backpropagation which tweaks the different weights and biases.
The backpropagation is done with gradient descent and there are two extreme version of these, either batch gradient descent (BGD) or stochastic gradient descent (SGD). What differs these two apart is how many samples are looked upon when doing the calculation for one update. BGD uses every sample at once, meanwhile SGD uses only one sample at a time. What usually happens is that BGD could get stuck in a local minima while SGD gets stuck in saddle points. Instead of using one of the extremes, mini-batch gradient descent could be used, which is something in-between. Mini-batch gradient descent uses fewer samples than all and more samples than one and updates every n training samples.

3.3 Architectures of deep neural networks

This section will explain some of the different architectures that exist in deep machine learning.

3.3.1 Recurrent Neural networks(RNN) and Long-short term memory (LSTM)

Recurrent neural networks (RNN) are used for sequential data such as $x^1, x^2, ...x^t$ which could be sentences, words and other time dependent data. Instead of sending

the whole sentence into the network at once, it sends one word each time and in the order they come in the sentence. Also the output of the first word is fed back into the network with the second word and then the output from the second and first word is fed back in again with the third word and so on. Figure 3.4 shows how RNN works. Where x is the input, h is the activation function, U, V, W are weights and o is the output [7].

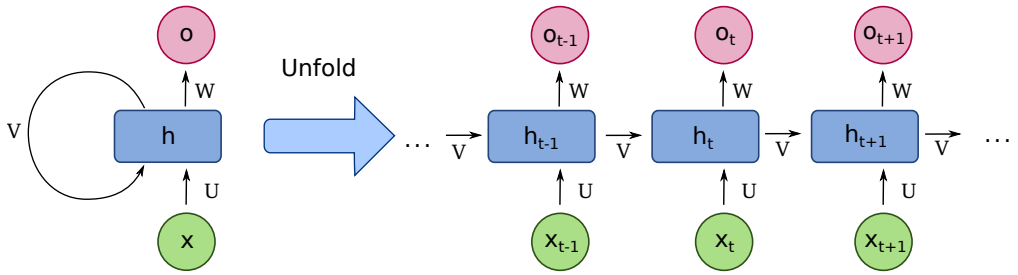


Figure 3.4: Reproduced recurrent neural network by François Deloche licensed under Creative Commons BY-SA 4.0 [2]

Although RNN is very good at handling sequential problems, it suffers from something called vanishing gradient descent. The issue is that when there is a very long sequence we unroll the RNN several times and get a very deep network. Then when backpropagation is done from the n layer to $n-1$ and the gradient is already very small for the n layer, we tend to get an even smaller gradient for layer $n-1$ and soon the gradient will be very close to zero. This is not only for RNNs though but for all deep networks, but more common here [13].

To deal with this problem LSTM were introduced [8]. They work as an RNN but have cell state, input, output and forget gates. Mathematically they are expressed as Equation 3.2, where $\odot$ refers to the element-wise multiplication, and visually as Figure 3.5.

$$\begin{aligned}
\text{forget gate, } f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
\text{input gate, } i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
\text{output gate, } o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\text{cell state, } c_t &= f_t \odot c_t - 1 + i_t \odot \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
\text{hidden state, } h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.2}$$

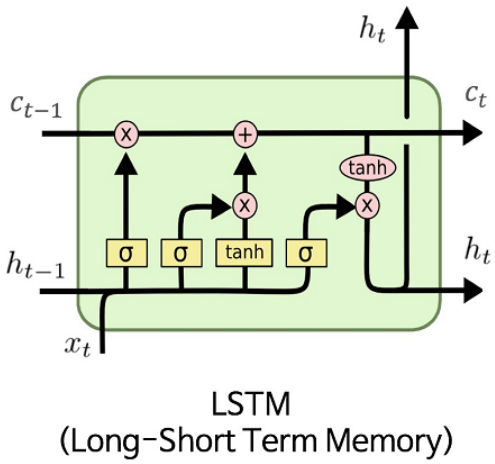


Figure 3.5: Reproduced LSTM cell by MingxianLin licensed under Creative Commons BY-SA 4.0 [1]

3.3.2 Convolutional Neural Networks (CNN)

Convolutional networks are usually used for 2-dimensional input, such as pictures, where they are able to extract high level features and then for instance deciding what type of objects there are in the picture [15]. However recently CNN has also been used for text and a CNN was built to be able to classify medical text [10]. It works such that a kernel, which is usually a smaller 2D-grid than the input, x, slides over the input and multiplies each overlapping input with the weights, w, and then sums them up. In Figure 3.6 there is an example where the input is 4x4 and the kernel is 2x2 and the stride, the amount of rows and columns the kernel is sliding, is 1x1.

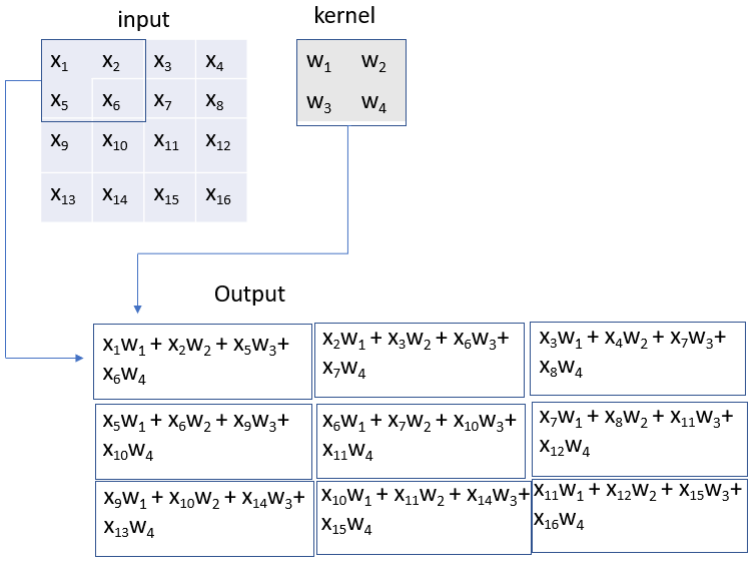


Figure 3.6: How convolutional network works

Depending on how the kernel, also known as filter, looks like there are many pos-

sibilities to extract different features. Imagine a filter that is 8x8 and has only 1 along the long diagonals and 0 elsewhere. This would result in finding X:s from the input.
Although CNNs are good with images there has been research with them in solving NLP problems as well. A comparative study with CNN and RNNs show that CNNs can outperform RNNs in some challenging NLP tasks but the study also shows that the CNNs is not far from the RNNs results in the other tasks [21].

3.3.3 LSTM-CNN

LSTM-CNN can be realized in two ways, either first the CNN-layer and then the LSTM-layer or the other way around. In the former case all of the input has to have a fixed length. Since in the case where the kernel slides over the matrix and the rows are of different length the convolution would not make sense. Then the input either has to be padded or truncated so that every input has the same length. The downside of truncating is that it removes information and the downside of padding is that noise is added. However in the case where truncation or padding is not needed, the CNN could first extract features before getting passed on to the LSTM-layer as in [9]. In the later case, the LSTM comes first and then the CNN. The upside of this is that there is no need of truncation or padding since a LSTM can handle variable input, therefore all the information is kept and no noise is added. The downside is that the intermediate results can not be used, since CNN needs a fixed input, and therefore only the last result from the LSTM is passed on. However all the intermediate results is encapsulated in the last and therefore some information about the time sequence is kept.
Many natural language processing (NLP) tasks has been performed with LSTM [18] [8] [19] but recently CNNs has also been used for classification of medical texts [10]. The combination of CNN and LSTM has also been used for forecasting [9], abstractive summarization of texts [3] and sentence unit detection [4]. Although the combination of LSTM and CNN has been used, it is always used in the same order, with the CNN first and then the LSTM. This is different from the model considered in this project, where the LSTM comes first and then the CNN. The reason for choosing this LSTM then CNN combination is to be able to keep the variable input length without having to pad every sentence and avoiding unnecessary noise.

3.4 Pre-processing

This section will present some pre-processing techniques which are used for simplifying the language, give more grammatical insight by labelling words in a category such as, noun, verb, etc. and it will also present an algorithm which makes words into vectors.

3.4.1 Lemmatisation,Stemming and Part-of-speech tagging

Lemmatisation and stemming are very similar to each other. The first one tries to find the base form of every word and at the same time looking at the context to get

a higher precision meanwhile stemming just takes the base form of the word without looking at the context. The difference could be seen in the following example: "Jag får äta får" ("I can eat sheeps"). Where the lemmatisation would change the first "får" to the base form "få" and do nothing with the second one, meanwhile stemming would either change non of them or both of them to "få". Both of these methods could be used as a reasonable pre-processing step to the deep learning network to simplify the language.
POS tagging is a method which tags each and every word depending on their corresponding grammatical category such as noun, verb, object, adverb etc. However it is not so easy to just have a list of every word with their corresponding POS since some words can have a different meaning under different contexts. This method might be very interesting since if it is possible to tag which is the object in the crash then it would be easy to say which vehicle that might have been the cause of the crash.

3.5 Word embedding

Word2vec is a word embedding algorithm, which means that it translate words into vectors [11]. It could utilize two different models, either CBOW (Continuous bag-of-word model) or skip-gram. In the CBOW architecture it uses the surrounding words to predict the current word. In the skip-gram architecture it uses the current word for predicting words surrounding it. In this project, Word2vec was retrained using CBOW model with the all accidents which were available, which was a total of 280354 crashes, with the dimensionality as 64, window of 7 and the rest of parameters as default. Later the window was changed to 3 to avoid including so much noise. Another worthy mention of word embedding algorithm is BERT [6], but since the existence of BERT was only found quite late in the project, Word2vec was the chosen one.

4
Methods

This chapter will explain what kind of neural networks that were used, how the experiments were conducted and also how the evaluation of the models were made.

4.1 Networks

Two networks were built, one with LSTM as its core and another one with LSTM-CNN as its core. In the LSTM-CNN, the LSTM layer was first to keep the variability of the input length. The LSTM-CNN network is shown in Figure 4.1. The complete network with the adding of pseudo-labels could be seen in Figure 3.1 where the learner is replaced by the network.

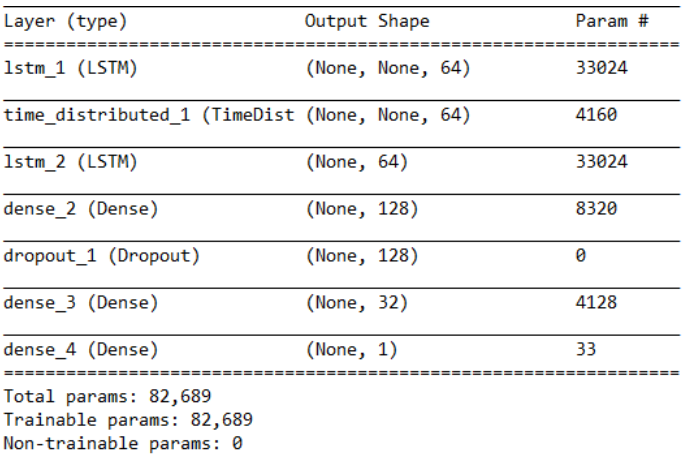
Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 16)	5184
time_distributed_1 (TimeDist	(None, None, 32)	544
lstm_2 (LSTM)	(None, 64)	24832
reshape_1 (Reshape)	(None, 8, 8)	0
conv1d_1 (Conv1D)	(None, 7, 32)	544
max_pooling1d_1 (MaxPooling1	(None, 2, 32)	0
flatten_1 (Flatten)	(None, 64)	0
dense_2 (Dense)	(None, 128)	8320
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 16)	2064
dense_4 (Dense)	(None, 1)	17

Total params: 41,505
Trainable params: 41,505
Non-trainable params: 0

Figure 4.1: The different layers of the LSTM-CNN network.

The LSTM-model looked as Figure 4.2 and has almost double of parameters compared to the LSTM-CNN. The style is very similar to the LSTM-CNN but the

convolutional layer and pooling layer are dropped.



Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, None, 64)	33024
time_distributed_1 (TimeDist	(None, None, 64)	4160
lstm_2 (LSTM)	(None, 64)	33024
dense_2 (Dense)	(None, 128)	8320
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 32)	4128
dense_4 (Dense)	(None, 1)	33

Total params: 82,689
Trainable params: 82,689
Non-trainable params: 0

Figure 4.2: The different layers of the LSTM network.

4.2 Experiments and Evaluation

A series of experiments were conducted as an iterative process of tweaking different parameters. The parameters can be seen in Table 4.1.

Table 4.1: The parameters which were tweaked

Parameter name	Description
batch size	How many samples trained simultaneously
add size	The amount of unlabelled data to be predicted on each training run
epochs	How many times to process the data
optimizer	An algorithm which tries to optimize the weights
learning rate	How fast the network learns
training size	The amount of labelled and unlabelled data that should be processed

However not all combinations were tested since there is an infinite amount of combinations. Instead one parameter was changed at a time while keeping the others constant. When a reasonable good value for a particular parameter was found it was kept constant and another parameter changed.
The loss was not a good measurement of how well a model did since the network labels things itself and could possibly be bad or good. Therefore the test data were used to measure the performance of the generated models. The metrics were Accuracy, Precision and Recall.
Accuracy is how many correct guesses that are made over all the test data and is describe in 4.1. If the actual label is true it means that the actual label is class 1 (e.g., vehicle 1 is in the front) and if the actual label is false it means that the actual label is class 0 (e.g., vehicle 1 is not in the front). TP is True Positive, TN is True

Negative, FP is False Positive and FN is False Negative. True and false refers to the actual label and positive and negative refers to the predicted label done by the model. It can be seen in Figure 4.3 how it works.

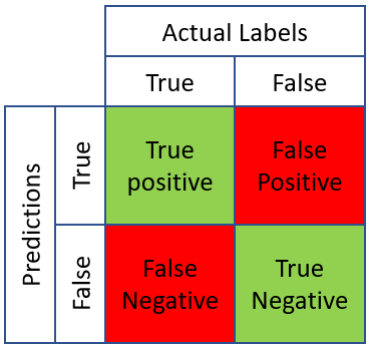


Figure 4.3: Overview of how True Positive, True Negative, False Positive and False Negative is connected. Horizontally is the prediction from the model and vertically is the actual label.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision is how many correct guesses out of all that should be positive values and is described in Equation 4.2.

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

Lastly, Recall which measures how the models does in predicting positive values and is describe in Equation 4.3.

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

Since the output from a model is a probability rather than 0 or 1, the accuracy was measured by looking at the predictions from a model and if the model made a prediction above or equal to 0.5 it was assumed that it guessed 1 and if it was below 0.5 it guessed 0. At later stages the implementation of the self-training change and added something called *low confidence* set. Which means whenever a model was uncertain about its prediction of a sample, it would go into a *low confidence* set. All models got not only to predict on the unlabelled data but also on the *low confidence* set every training run. The changes can be seen in Figure 4.4.

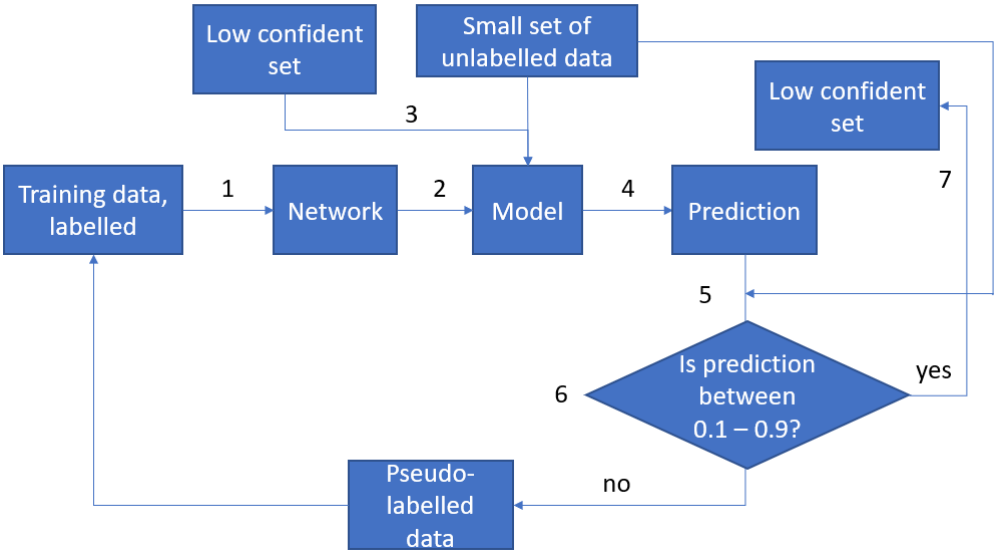


Figure 4.4: How a semi-supervised learning network works: 1. Training data is sent to the learner 2. network creates a model 3. A small set of unlabelled data and the whole set of low confident is sent to the model 4. The model makes predictions on both of the sets 5. The unlabelled data and the predictions creates a pseudo-labelled data set and the low confident samples get predicted again on 6. If a prediction is between 0.1 - 0.9 it's added to the low confident set, otherwise it becomes pseudo-labelled data and added to the training data

4.2.1 Resources

All the code to extract the data was written in Python 2.7 and all the code for the models were written in Python 3.6. Chalmers also provided the project with a server called Bayes and the server is also the reason why the code for the models are written in version 3.6. Since the server had many users each user could only use one CPU and one GPU at a time, meaning maximum of two tests could run simultaneously. One test took roughly 1 day.

5
Results

This chapter will present some of the key results. Both architectures are using default parameters which can be seen in Table 5.1. If there is nothing mentioned about the parameter then it is assumed that the default one is used. The results are presented by firstly explaning what kind of experiment that was made, secondly, a graph to show the results and lastly, what the graph is showing.

5.1 LSTM-CNN results

This section will present the results for the LSTM-CNN model. Loss was used as a performance measure in the first results (Figure 5.1). The parameters changed were the add size and the epoch. The training only consisted of 100 labelled data and a training size of 5000 samples.
In Figure 5.1 the trend for the blue and orange looks similar and has probably converged. The green is unclear how it will develop and the red line has an ascending trend and not yet converged. However comparing the green, orange and blue line it is clear that the lower add size the lower loss. Similar with the red and green line, which compares epoch, the higher the epoch the lower the loss.
In Figure 5.2 accuracy is used as a measurement and on only 20 test samples, with a few different models. The models were trained on 100 labelled samples.
In Figure 5.2 the blue line is clearly ahead of the others at around 0.95 accuracy. The orange line, with only epoch of 5 has an accuracy around 0.7 to 0.8. Similar pattern for the purple and red line with *add size* as 8 and epoch of 5 and 64. However they differ in *training size* which indicates that epochs are more important than training size, although it is a very small test set with only 20 samples. The test set had an average length of 8 words per sample, where the longest sentence was 16 and the shortest was 5. The average length for a sample with 2500 training samples was

Table 5.1: The default value of the parameters

Parameter name	**Value**
batch size	32
add size	128
epochs	5
optimizer	Adam
learning rate	0.001
training size	Differs and will be mentioned

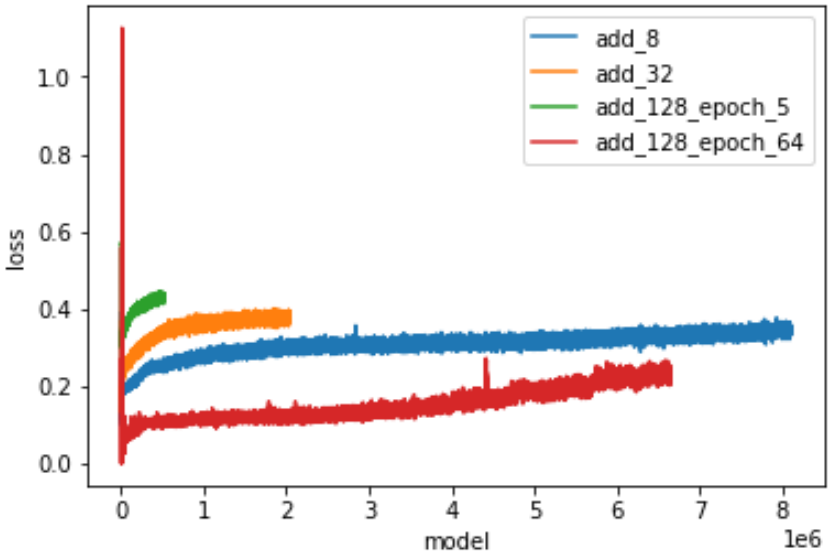


Figure 5.1: This graph shows the loss of the network with *add size* as, 8, 32 and 128, but also *add size* 128 with both *epoch* 5 and 64. The y-axis is the loss and the x-axis is *training run* · *epoch*. The *training size* was 5000.

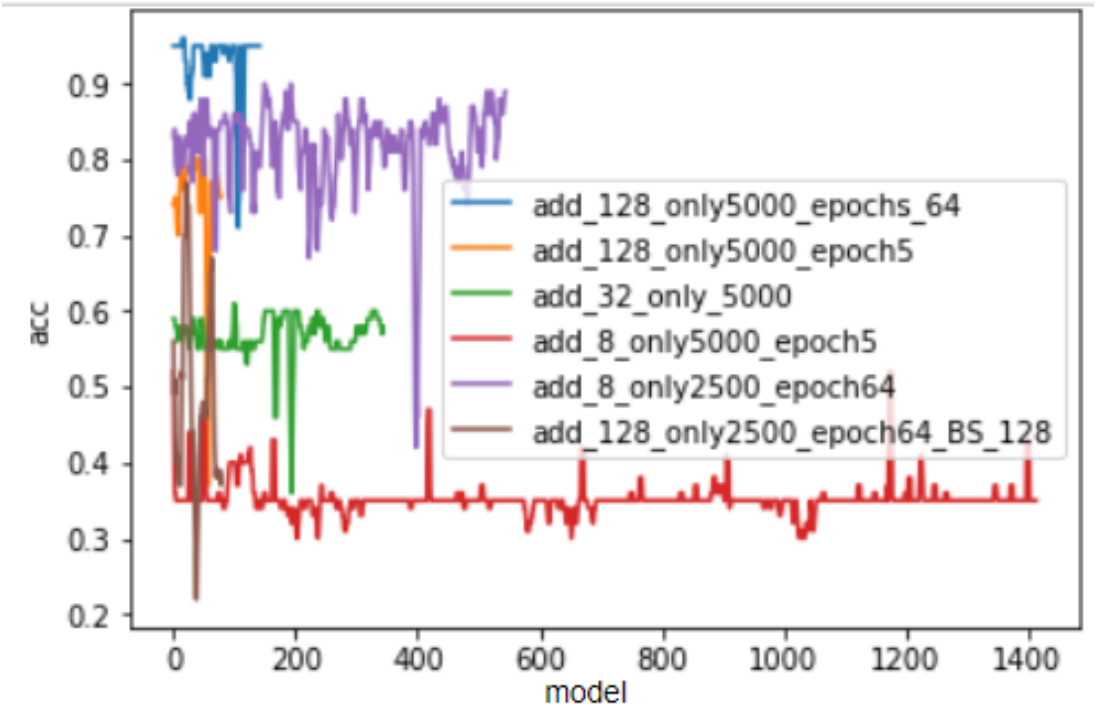


Figure 5.2: Shows the accuracy for 20 test samples. These samples averaged less than 8 words per sample, which is very short.

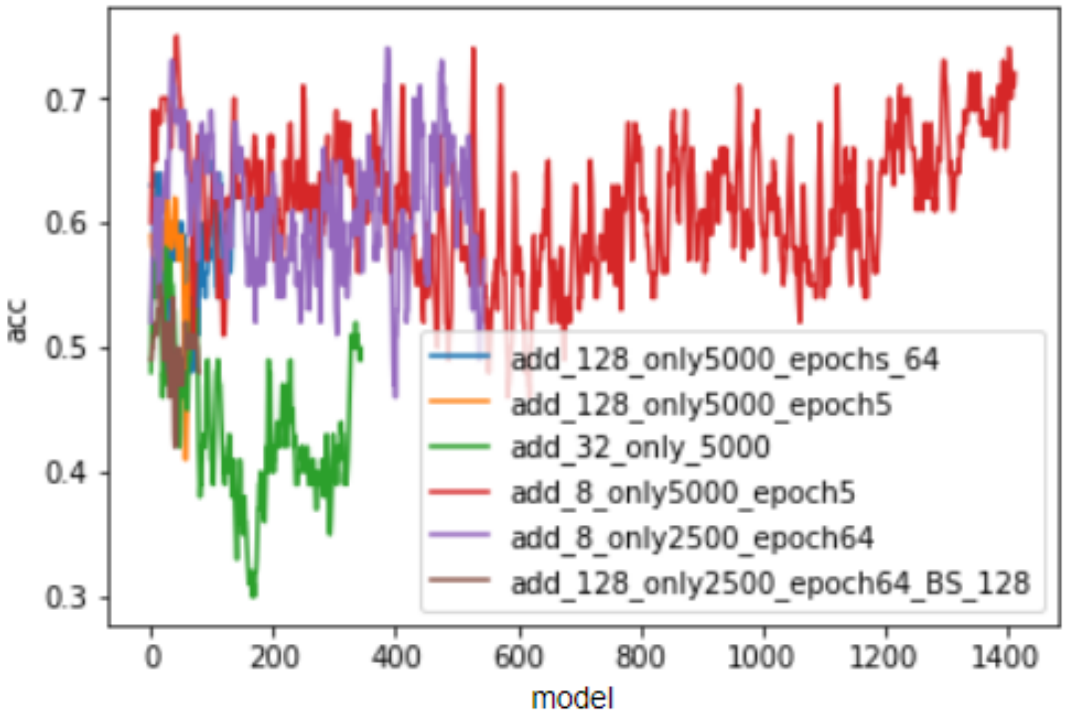


Figure 5.3: In this graph there were 20 random test samples. The average length of a sentence was 29 words.

19.5 and with 5000 training samples was 17.6. This is the same for all models with the same training size.
Figure 5.3 uses the same models as Figure 5.2, and the only difference is a very different test set. Still only 20 samples, but the average length of a sentence is 29 words, shortest sentence 11 words and longest 52 words. The big difference in the test sets might also be a reason why the results differ so much, since the models have not seen many sentences around 29 words, meanwhile they have covered sentences longer than 7.
The blue line which was clearly the best in Figure 5.2 resides around 0.6 in accuracy now which is almost the same as the orange line. The red and purple lines are quite similar in the start however the red line ends with a peak of 0.7 in accuracy. This contradicts the results from Figure 5.2 which gives a slight indication that the length of the samples in the training set are important or the importance of seeing many samples (e.g, having a larger training set, 5000 is better than 2500) rather than training on same samples many times.
The next experiment (see Figure 5.4) used accuracy instead of loss and has similar comparisons as Figure 5.1. The models were trained with 5000 training data, 2500 training and also started out with 100 labelled data. The test data was only 60 samples.
The blue line with add size of 128 and epoch of 64 got the highest accuracy and trailing not far behind was the purple line with add size of 8 and epoch of 64. Compare to Figure 5.1 where it was clear a lower add size was good and higher epoch was good, this graph contradicts that based on the blue and purple line.

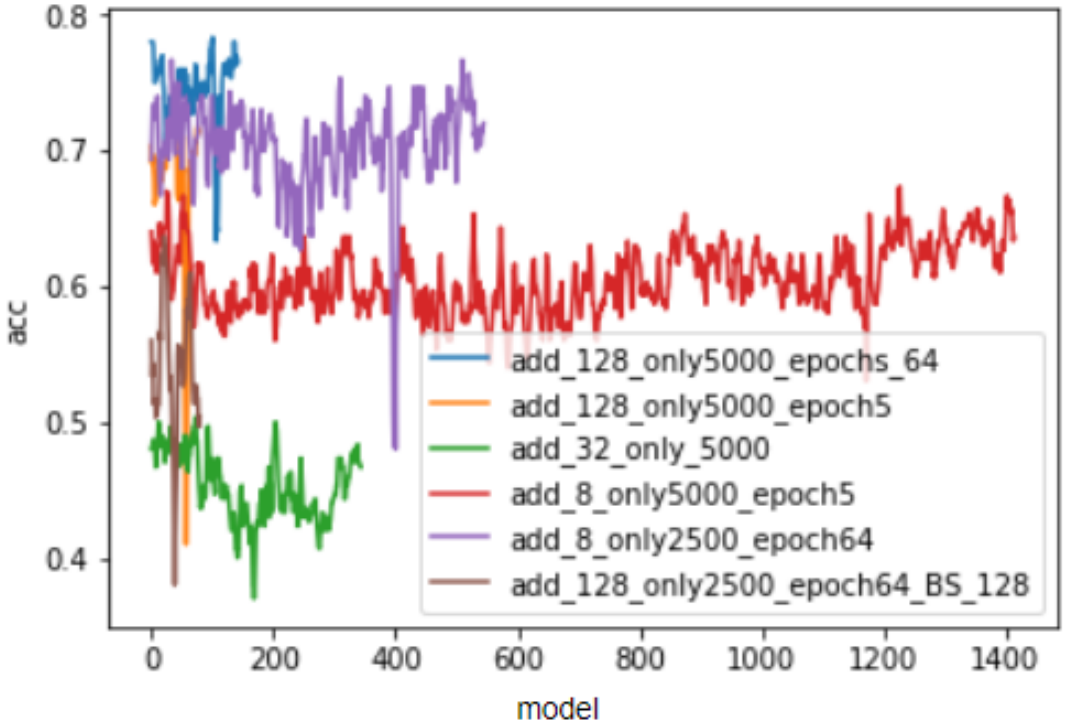


Figure 5.4: Models with different epochs, add size, training size and batch size tested on 60 samples.

Comparing the brown and blue lines which have different training sizes, 2500 vs 5000 respectively, and different batch sizes, 128 vs 32 (default value of batch size), there is a clear performance gap from 0.55 to 0.77.
The same experiment as in Figure 5.4 got retrained with a starting golden set of 160 samples. The models were then tested on a larger test set containing 347 samples. The reason for the test set being much larger than the training set was to see if the models were consistent with a larger test. The results are shown in Figure 5.5.
Compared to Figure 5.4 where the blue line was at the top with 0.77 accuracy and in Figure 5.5 around 0.66 in accuracy, the purple line which has add size of 8 and epoch of 64, is at the top in Figure 5.5 with similar accuracy as in Figure 5.4. Other than that the models still have similar performance but with a larger test set.
The next experiment (shown in Figure 5.6) tested a modified version of self-training (i.e., the one using low confident set shown in Figure 4.4). The training set consists of 507 labelled training data and a test set of 86 samples. It was also tested with a low epoch size of only 2 and a high training size of the whole unlabelled set.
Compared to Figure 5.4 and 5.5 there is a slight boost in accuracy reaching around 0.80 in Figure 5.6 instead of below 0.80. The blue and orange lines have very different add sizes as well, blue line with 64 and orange with 1024, but still gets similar performances.
Figure 5.7 also used the low confident set, with 507 labelled training data and a test set of 86 samples. It tested two different add sizes of 8 and 128 with training size of 2500 and 5000 respectively.
In Figure 5.7 both lines have a stable trend. The orange lies around 0.72 to 0.78

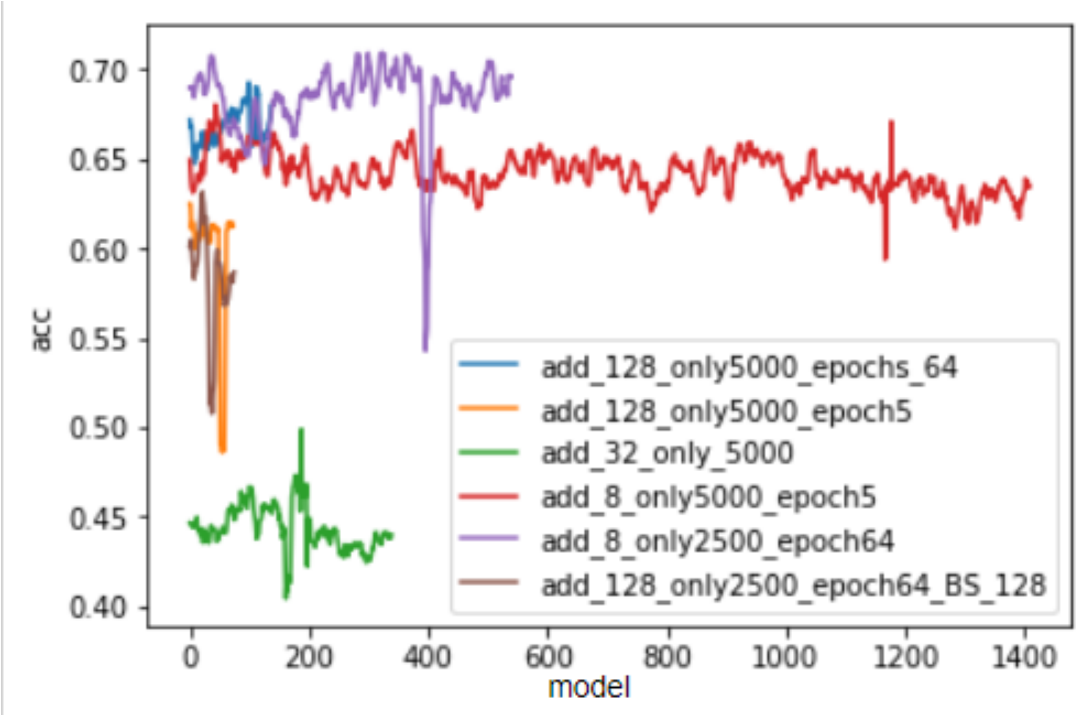


Figure 5.5: Similar experiment as in Figure 5.4, but with a larger training set of 160 samples and a larger test set of 347 samples.

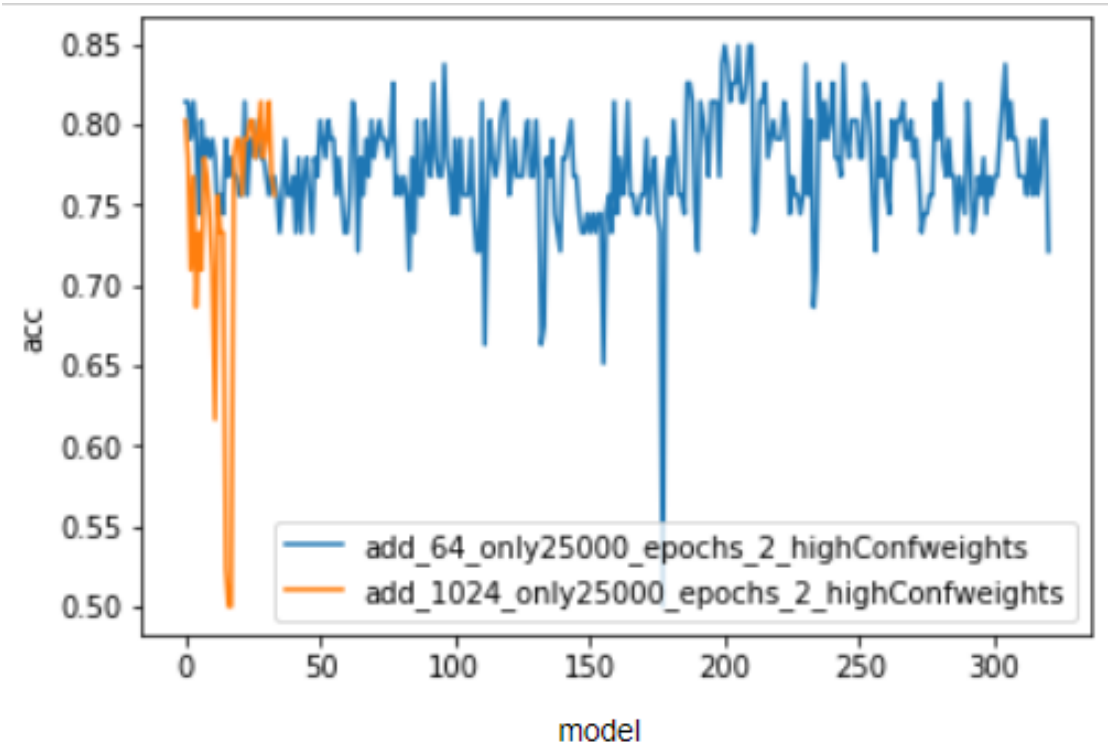


Figure 5.6: Accuracy test with high confident test and only 2 epochs but trained on all the data, but also 507 initial training data and 86 test data.

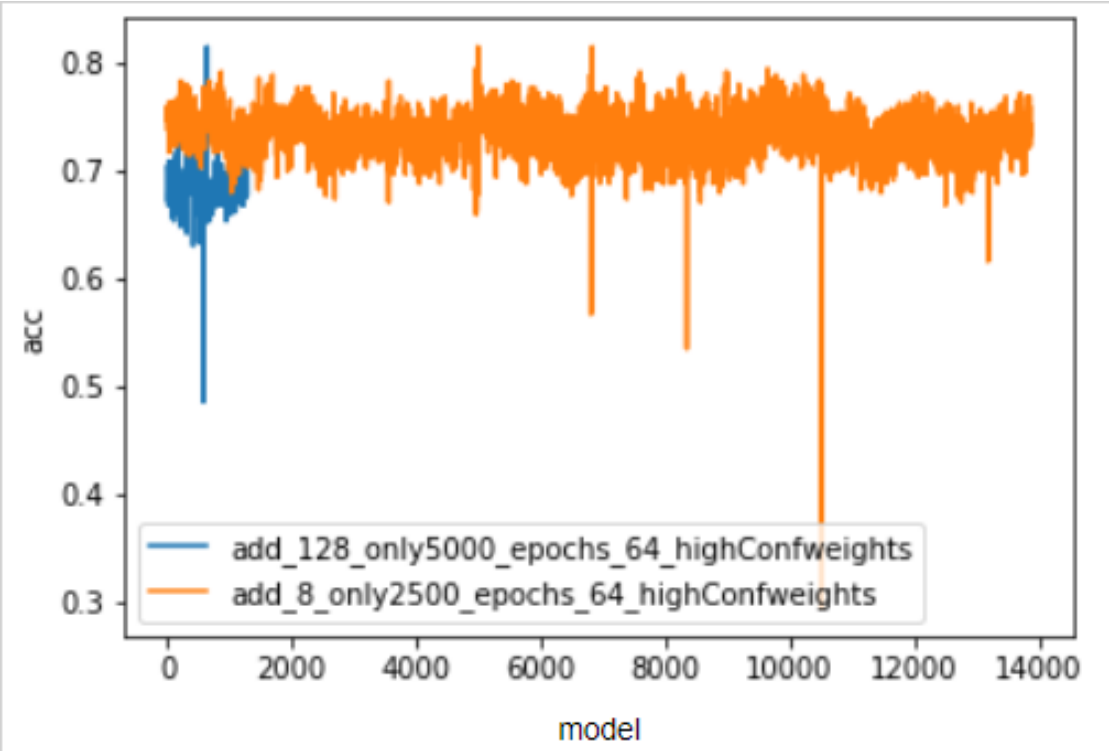


Figure 5.7: Accuracy test with high confident test.

and the blue around 0.66 to 0.71. Compared to earlier graphs which did not use low confident, the trend is much more stable. This is probably because of the low confident test which forces the model to be certain before adding an unlabelled sample to the training set. Using an epoch of 64 and low confident could also make the models very much the same since the same samples are iterated over several times and could also be a reason in the stability.

Figure 5.8 used a different window size for the Word2vec. Instead of using the default window size of 7 it used 3. It also tested different amount of epochs to see how the epochs affected the accuracy.

The green line in Figure 5.8 has the highest accuracy and is very stable around 0.7 in accuracy. Both the orange and blue line however has interesting trends since both start off poorly and decreases but after a while they increase. However in the end both rapidly decreases again. The green line in Figure 5.8 was very stable and around 0.7 in accuracy and the test data was roughly divided 30/70 which was the reason for the next experiment. Figures 5.10 shows how the green line was predicting when vehicle 1 was not in the front and Figure 5.9 when vehicle 1 was in front.

Figure 5.9 shows that the accuracy is very close to 0 for predicting when vehicle 1 is in the front.

Figure 5.10 shows that the accuracy is very close to 1 for predicting when vehicle 1 is not in the front. Which shows that the green line in Figure 5.8 might be a bad model since it only predicts that vehicle 1 is not in the front.

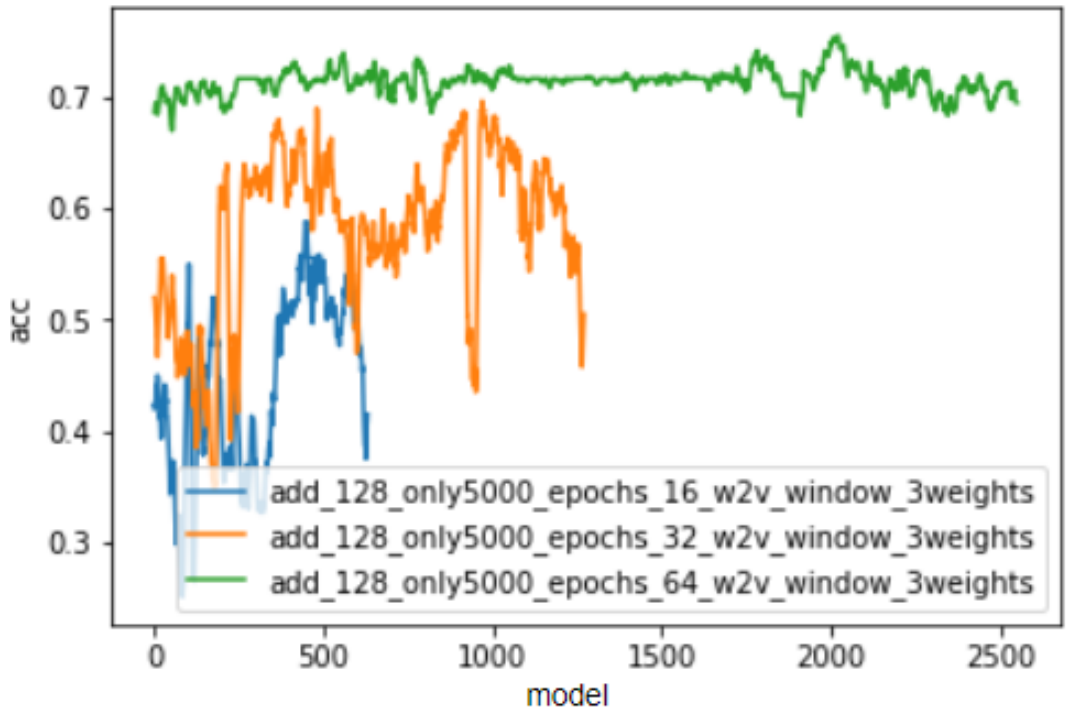


Figure 5.8: In this graph it was tested with a window length of 3 in word2vec and several different epochs.

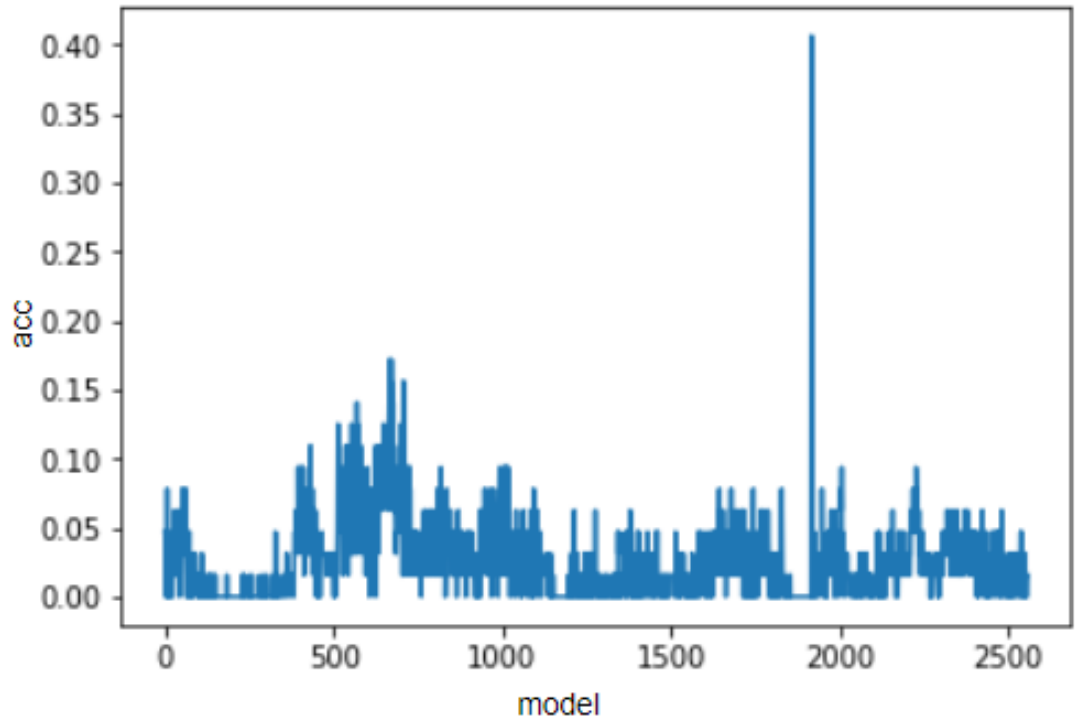


Figure 5.9: This graph shows how often the models are guessing correct when vehicle 1 is in the front.

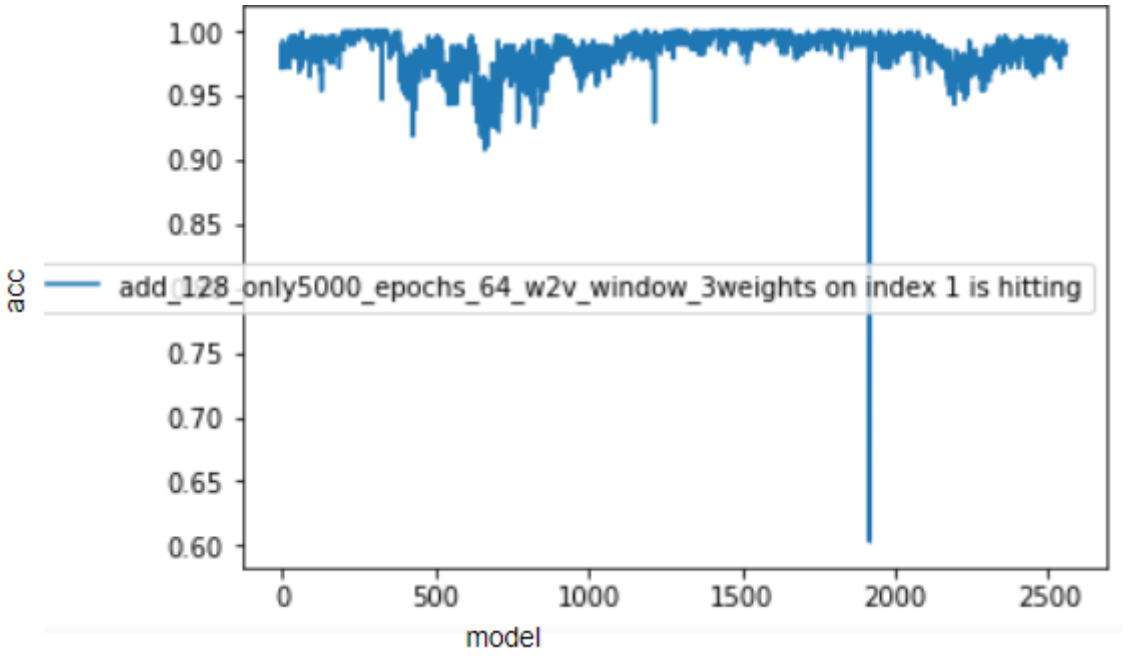


Figure 5.10: This graph shows how often the models are guessing correct when pb 1 is **not** in the front

5.2 LSTM results

This section presents the LSTM only results. It has the same default parameters as the LSTM-CNN and can be seen in Table 5.1. Also the LSTM uses only the changed version of the self-training with low confident set (see Figure 4.4). The models were also trained with 507 training data and tested on 86 samples. The test data was divided evenly between positive samples and negative samples.
This experiment (Figure 5.11) uses the whole unlabelled set as training size, batch size of 128 and compares two different add sizes of 128 and 256. It also shows the precision and recall for these models.
In Figure 5.11 the blue and orange line have very similar performance, around 0.80 in accuracy and 0.6 in recall. However the precision for the orange line is very close to 1, higher than the blue line.
The next experiment compared the batch sizes of either using SGD (batch size of 1) or a batch size of 512. Both models used the whole unlabelled set and also had 507 labelled as training set. The accuracy was measured on 86 test samples.
In Figure 5.12 the blue line, which is using SGD has a slightly higher accuracy than the orange line. The trends of both curves are similar and there is just a small difference in the amplitude. The accuracy is around 0.75 and compared to Figure 5.11 this is slightly worse.
The results shown in Figure 5.13 tested different learning rates instead of the default value of 0.001. The learning rates tested were 0.1 and 0.01. Both models used 507 training data, the complete set of unlabelled data and tested on 86 samples. It used high confidence during training, i.e., that if the confidence for a model was between 0.1 to 0.9 for a test sample then this sample was added to a set called low confident

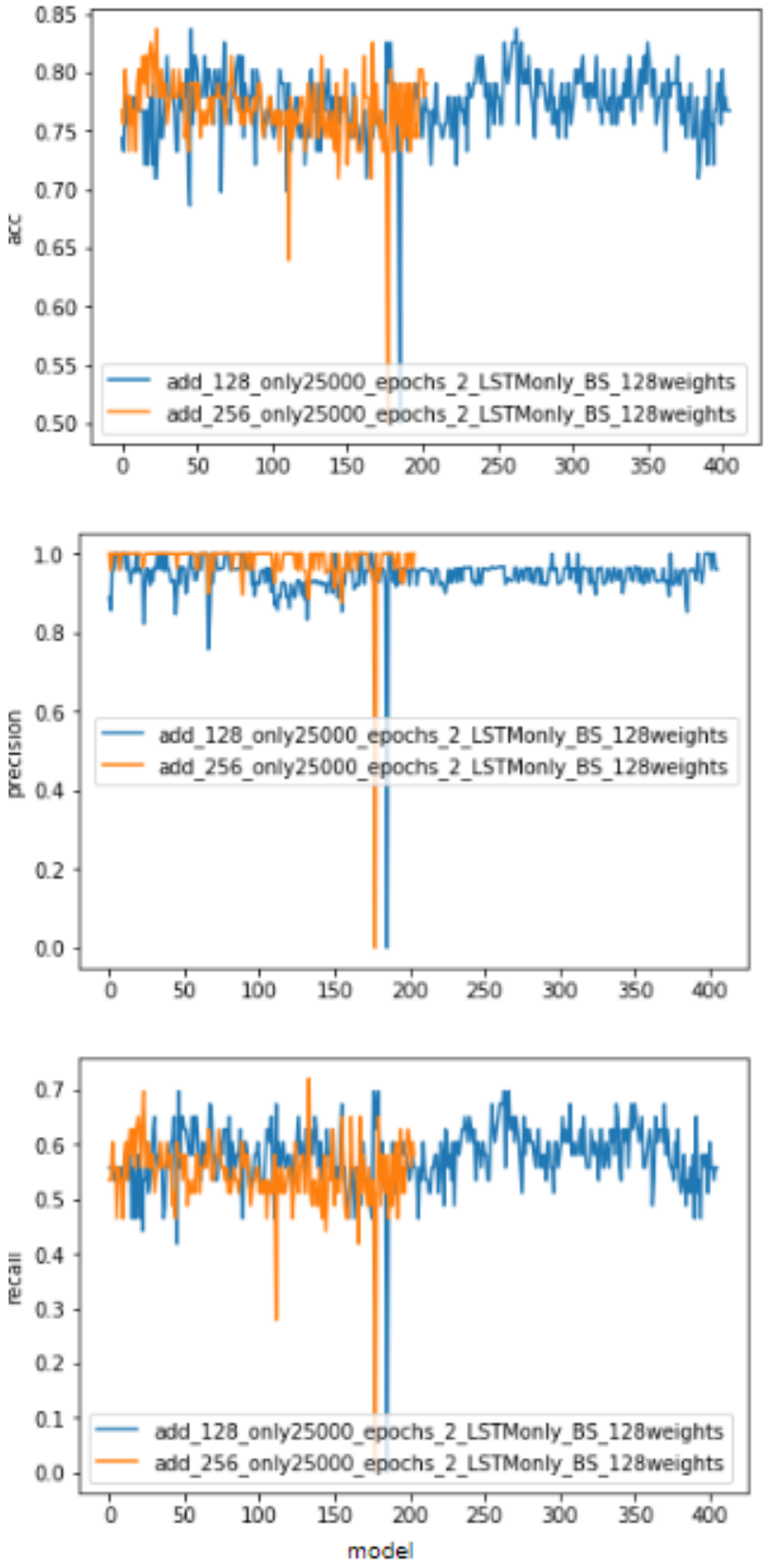


Figure 5.11: These graphs show the accuracy, precision and recall for the LSTM network.

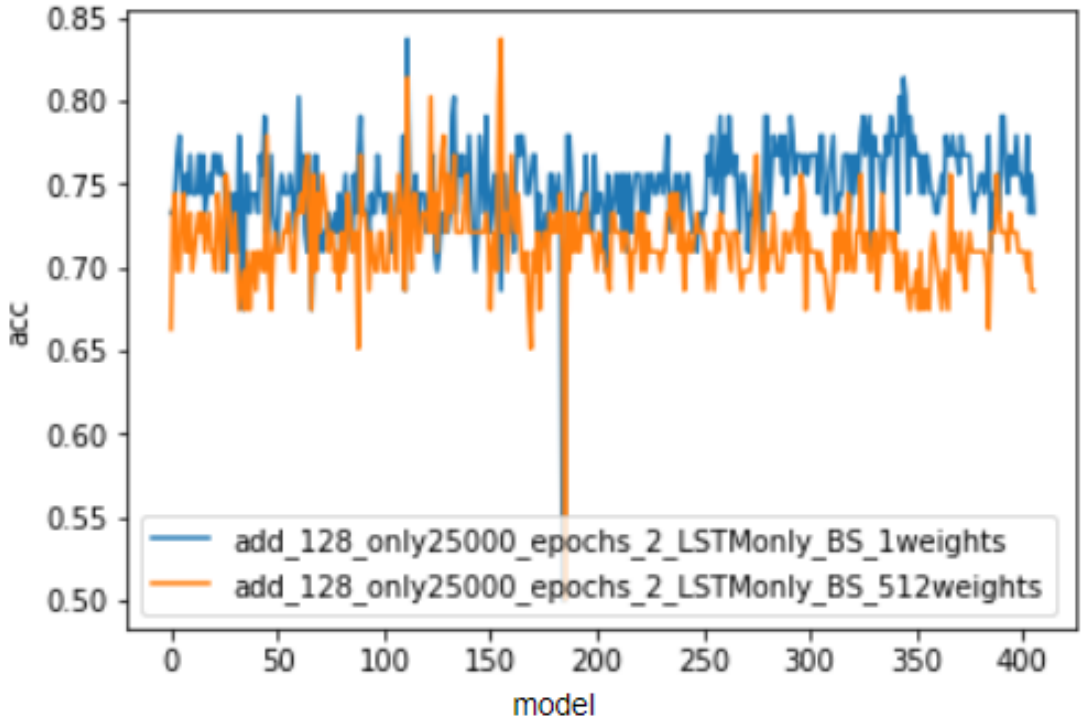


Figure 5.12: In this graph it is shown the comparison between Stochastic gradient descent ($batch\ size = 1$) and mini batch gradient descent ($batch\ size = 1$.

instead of being added to the training set, and used a test set which was divided between 50 and 50.
In Figure 5.13 the accuracy is exactly 0.50 for both the models. Both models also have a recall of 0.00 which indicates that the models are predicting same value all the time.

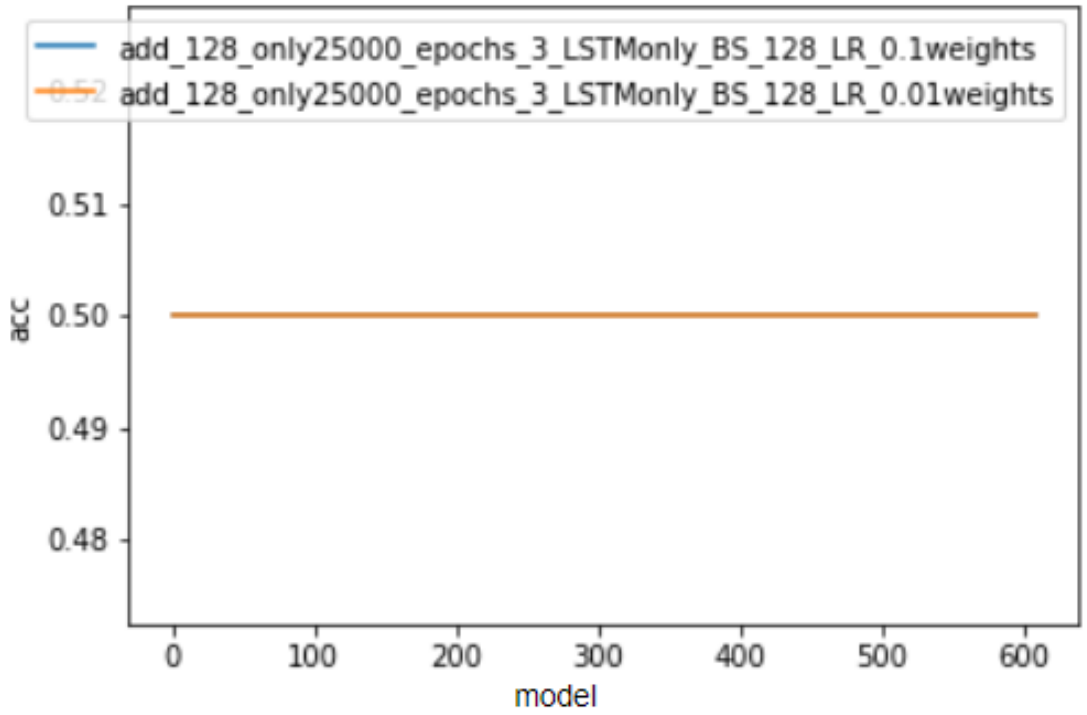


Figure 5.13: In this graph the accuracy for learning rates of 0.1 and 0.01 are shown.

6
Discussion

This chapter will discuss the results, but also insights about semi-supervised learning and the data that has been used. It will also discuss issues using the data in both the ethical and practical aspect.

6.1 Evaluation of the results

There has been many models produced, however understanding and evaluating if the models have been good or not has been difficult. One reason is that there is no other similar work or benchmarks to compare with. Also the lack of test and training data made it difficult to really trust the results. In the end there was only 507 training data and 86 test data and the 86 test data are too few to cover all the characteristics there are to write the same thing. Another thing which made it hard to evaluate the models is that the data continuously grew during the work, which means all models have not had the same initial training data, nor the same training sizes. Which means the comparisons between models are not fair.
In Figure 5.1 it is clear to see that a lower add size yields lower loss. The reason for this is because it trains much slower and has more time to overfit to the data. The number of training runs could be calculated by $\frac{trainingsize}{addsize}$ which means that using an add size of 8 instead of 128 will have 16 times more training, similar reason for the **epoch**. Although loss is an easy metric to look at, it is not reliable in this setting, because if the networks start off with predicting wrong labels and adds them to the labelled set the labelled set will quickly be diluted from the truth and the accuracy can be poor but the loss can still be low. Looking at Figure 5.1 which compares loss and Figure 5.4 which uses accuracy as measurement the results do not overlap and contradicts each other. A low loss should yield a high accuracy and vice versa. This means that the accuracy and loss are not corresponding to each other and accuracy should be the preferred metric based on the reliability of the loss.
Figure 5.2 and Figure 5.3 utilizes the exact same models, but different tests. One of the tests had 20 short samples (the samples averaged less than 8 words per sample) meanwhile the other had long ones (the samples averaged 29 words per sample). This significant difference also yielded a very different result, where the best model tested on the short test had an accuracy around 0.9 meanwhile the best model on the longer one resided around 0.65. This gives the indication that the length of the samples are of importance. This could also mean that the models only learn to evaluate sentences up to a certain length and can not make general conclusions about the data. Another reason for this behaviour is that the model has not seen

enough data of this kind to evaluate it correctly.
Figure 5.8 tested shorter window length for word2vec and different epochs. It has a clear indication that the higher epoch the better the accuracy. It also reached a similar accuracy, around 0.7, as other experiments. However further investigation was made due to the fact that the data was divided roughly 30/70. In Figures 5.10 and 5.9 the classes was divided and each figure only shows the accuracy for one class. The accuracy for one class was close to 1 and for the other class close to 0, which means this model only predicts one class all the time. Which makes it a bad model.
Figure 5.7 had a stable trend and used high confident test. The reason for the stable trend is most likely because of the high confident test. If the high confident test filters out all the samples which the models are not good at and only keeps the ones which they are good at the models might train on only samples with the same characteristics. This will also make the models not good at generalizing.
Figure 5.6 was trained on 507 labelled samples but also got to see the whole unlabelled set. It also used a low epoch of only 2 and used the changed version of self-training with the low confident set. This yielded a higher accuracy than Figure 5.5 which was trained only on 160 samples, got to see at most 5000 unlabelled samples and had a higher epoch of 64. The reason for the out performance lies in the amount of data both tests got to see, but also in the epoch. A high epoch is not really needed since the network will see the training data several times already due to the self-training and it is better to see a variety of data.
Figure 5.11 showcases the LSTM model with training size of 25000 and epoch of 2. The model got an accuracy around 0.8, precision close to 1 and recall around 0.6. With such a high precision means that the model is good at predicting true labels when it should be true labels. However with a recall of 0.6 means that 40% of the time it predicts false to a true label. Comparing Figure 5.11 which uses only LSTM and Figure 5.6 which is a combination of LSTM and CNN the LSTM-CNN uses half of the parameters compared to the LSTM. This means that the LSTM-CNN trains faster than the LSTM only. Although with only 86 test samples it is difficult to draw any concrete conclusions.
Figure 5.12 compares the batch sizes and got around 0.75 in accuracy. However both the orange and blue lines have accuracy worse than Figure 5.11 orange and blue lines which uses batch size of 128. This indicates that a reasonable batch size is larger than 1 and smaller than 512.
Figure 5.13 tested higher learning rates than the default one (default learning rate was 0.001), 0.1 and 0.01. The accuracy for the models was 0.50 and both models had a recall of 0.00. Which means that both of the models always predicts class 0. This could mean two things, either the models are bad or that the threshold for being labelled as 0 should not be below 0.5 and to be labelled as 1 to be 0.5 or above. If the models have predictions close to 0.5 it means that the model is uncertain which class it should label. Also since the data seems to be unbalanced it makes sense to change the threshold.
The results give a clear indication that it is better to look at more data, rather than training on the same data over and over again. Also in the semi-supervised case loss and accuracy does not necessary correlate to each other and therefore accuracy

is the preferred measurement. Using the high confidence during training yielded higher accuracy and should be something that needs more investigation. Batch size should be kept low since there will already be a lot of training on the same sample due to the amount of training runs. Similar with the amount of epochs. Add size is a bit unclear, however there are some slight indication that it should be kept low. Lastly the LSTM-CNN network and LSTM only has similar accuracies however the LSTM-CNN used half of the parameters compared to the LSTM which means faster training and perhaps could be better with equal amount of parameters.

6.2 Thoughts on semi-supervised learning

Semi-supervised learning is an interesting technique trying to bypass the huge amount of labelled data one needs and the uncertainty in the unsupervised case when it comes to evaluation. However there are still many uncertainties and it is not as evolved as supervised learning. There are new parameters to tweak and many things to ask.

6.2.1 Epoch

One interesting parameter is the epoch. As shown by the results it might be a parameter you skip or setting it very low since the self-training makes the network train on the same samples over and over again anyways.

6.2.2 Add size

Another very special parameter is the add size. This parameter should be very crucial for how the networks performs since adding too many samples at once would most likely dilute the network and adding too few would make it overfit. If the network should process all the unlabelled data then the add size also determines the time it will take to train the network. The add size could also not be constant but perhaps change over time or change depending on how large the labelled set is.

6.2.3 Overfitting

Overfitting might be strange to discuss in semi-supervised learning since it is difficult to predict how well the network is doing. However as shown by many results the accuracy was around 0.7 and the distribution was 30/70 which was a bit suspicious in conjunction with the precision and the Figures 5.9 and 5.10.

6.2.4 How to add unlabelled to labelled

Another interesting part is during self-training, how to add the unlabelled to the labelled. It could be done as in this project i.e., letting the network predict and adding them with the labels assigned or making sure the network has a high confidence before adding them. There could also be other ways, like adding weights to the labelled ones. E.g. if the network starts out with 500 golden samples, and we

add 500 more then there could be a weight applied of 0.5 to every sample. Which means every sample gets a weight of $\frac{\#\text{initial samples}}{\#\text{labelled}}$. It could also be a decaying weight depending on for how long the samples have been in the labelled set, to avoid being trained on too many times.

6.3 Practical aspects with the data

The data is inconsistent, i.e., similar cases have different descriptions, cases without any index number (e.g., only using "car" rather than "car 1"). The data also consists of spelling errors and cases which are impossible to derive anything from. The labelled data is also unevenly distributed between the two groups, around 70% to 30%. Something which were researched around but not utilized in this project are lemmatisation, stemming and POS. All of these would be interesting to test, stemming and lemmatisation to simplify the language by changing words into its base form and with POS to make it more detailed. POS tagging would add useful grammatical information in what kind of word it is, noun, verb, adverb etc. It could also tag whether it is an object or a subject, which could be useful information. This could perhaps have changed how the network interpreted the sentences. Another interesting part was to play around with the word2vec parameters. Perhaps higher dimensionality, larger context window and maybe to train it on corpora outside the domain to get a better understanding of the language. It would also be interesting to try with a different model such as BERT which reads the sentences from left-to-right and right-to-left at the same time which gives a deeper understanding of the sentences. BERT is also pre-trained on many languages and there could be a continuation of training for the specific domain.

6.4 Ethical aspects with the data

Since the data is from STRADA and one has to apply to get access to the data it comes with some restrictions. It is not shareable, which means it can not be stored anywhere which makes it difficult to use server such as Google or Amazon for training the network. Another thing could be privacy issues. Even though there is no names connected to the data there are sometimes road names and vehicle types but also hospital reports. If this information gets leaked a person could possibly be portrayed, blackmailed or something else. The purpose of STRADA is to improve traffic safety and work towards the goal of zero vision (i.e., zero crashes in traffic). Therefore it is important that the privacy issues are respected; this influences the data sharing. Another thing about the data and the model is that one needs to be careful what question the model is answering. The network should predict a neutral answer based on the input data and not a biased or be able to stir emotions. For instance it is okay for the network to predict car 1 was in the front, but it is not okay to say car 1 was the responsible for the crash. The reason to have a neutral prediction is because then the accuracy does not matter too much. However if it is a biased prediction the accuracy is very important and should probably be 1. Imagine that this was developed more and used for insurance companies or fining, convicting

persons based on the crash. Then it is very important that the model does correct all the time otherwise it could fine or convict the wrong person. Another reason to have a neutral prediction is to protect the creator of the model. Otherwise if the model is predicting something bad, is it the creators fault then?

7
Conclusion

The purpose of this thesis was to implement an automatic system which could determine if a vehicle was in the front or not based on a text description. At this stage, the predictions made by the system or the models produced here (i.e., if the vehicle 1 is in front or not) cannot be reliably used. This is because there are still some uncertainties in the results. However, there has been a lot to learn from the journey to come to this stage. The journey is summarized briefly in this section. The data used was from STRADA and it was only the rear-end crashes with 2 vehicles that were used. The data went through some common pre-processing steps suchs a making everything lower case and removing punctuation. After that the word2vec model was retrained on all the cases in the STRADA database. Then two models were built, one called LSTM and the other LSTM-CNN. Both models had similar performance when the parameters were similar, but LSTM-CNN had half the parameters compared to the LSTM. Epoch should be kept low since the self-training makes the network see the samples several times anyway. Also it was better to see a lot of varied data, rather than the same data more often. add size was an unclear parameter and needs more testing to understand it better. There also needs more investigation on how to add to the labelled set, however adding only when the model has a high confidence gives a better performance than adding based only on the prediction from the model. Loss is probably a metric which should be avoided in semi-supervised learning due to the unreliability in the networks performance during training. The data was inconsistent and varied, since there existed cases with spelling errors, without index numbers and similar cases could have very different descriptions. The data also seem to have a distribution of 70/30. The data ended up in 507 training samples and 86 test samples. All in all the models reached around 0.8 accuracy which could be used for future benchmarks.

7.1 Future work

Future work could look at how to add from the unlabelled set to the labelled one. Implementing weights could be one strategy, where there are several ways to do it. Another thing would be to look at the add size parameter more and perhaps instead of adding a constant number each training run it might be better to have it change during the training. To do these things a lot of testing needs to be done. Another thing could be to look at the sketches of the crashes and do both prediction of the sketches and the text and then perhaps merge the results. There could also be more things done revolving around the language, such as using lemmatisation, stemming

or part-of-speech tagging. Testing different word embedding models, with different parameters would also be useful for the overall performance.

Bibliography

[1] Lstm. https://upload.wikimedia.org/wikipedia/commons/9/98/LSTM.png. Accessed : 2019-06-20.

[2] Recurrent neural network. https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg. Accessed : 2019-06-20.

[3] Santa Barbara. Abstractive Text Summarization Using Hierarchical Reinforcement Learning. pages 857–875, 2018.

[4] Duy Cat Can, Thi Nga Ho, and Eng Siong Chng. A Hybrid Deep Learning Architecture for Sentence Unit Detection. *Proceedings of the 2018 International Conference on Asian Language Processing, IALP 2018*, pages 129–132, 2019.

[5] Susantha Chandraratna and Nikiforos Stamatiadis. Quasi-induced exposure method: Evaluation of not-at-fault assumption. *Accident Analysis and Prevention*, 41(2):308–313, 2009.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. http://arxiv.org/abs/1810.04805 Accessed : 2019-06-20.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org, Accessed : 2019-04-15.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[9] Chiou Jye Huang and Ping Huan Kuo. A deep cnn-lstm model for particulate matter (Pm2.5) forecasting in smart cities. *Sensors (Switzerland)*, 18(7), 2018.

[10] Mark Hughes, Irene Li, Spyros Kotoulas, and Toyotaro Suzumura. Medical Text Classification Using Convolutional Neural Networks. *Studies in Health Technology and Informatics*, 235:246–250, 2017.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. Technical report, 2013. http://arxiv.org/abs/1301.3781, Accessed : 2019-04-15.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

[13] Michael A. Nielsen. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/chap5.html Accessed : 2019-06-20.

[14] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D Cubuk, and Ian J Goodfellow. Realistic Evaluation of Deep Semi-Supervised Learning Algorithms. (NeurIPS), 2018.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[16] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Mutual Exclusivity Loss for Semi-Supervised Deep Learning. *2016 IEEE International Conference on Image Processing (ICIP)*, 2016-Augus(June):1908–1912, jun 2016.

[17] Karen Simonyan and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. pages 1–14. ICLR, 2015.

[18] Martin Sundermeyer, Ralf Schlueter, and Hermann Ney. LSTM Neural Networks for Language Modeling. *Proceedings of INTERSPEECH*, pages 194–197, 2012.

[19] Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. A Unified Tagging Solution: Bidirectional LSTM Recurrent Neural Network with Word Embedding. 2015. `http://arxiv.org/abs/1511.00215`, Accessed : 2019-04-15.

[20] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. pages 1–18, jun 2019. `http://arxiv.org/abs/1906.08237`, Accessed : 2019-04-15.

[21] Wenpeng Yin, Katharina Kann, Mo Yu, Hinrich SchützeSch, and Lmu Munich. Comparative Study of CNN and RNN for Natural Language Processing. Technical report, 2017. `https://arxiv.org/pdf/1702.01923.pdf`, Accessed : 2019-04-15.

[22] Xiaojin Zhu and Andrew B. Goldberg. Introduction to Semi-Supervised Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 3(1):1–130, jan 2009.