City University of London

School of Mathematics, Computer Science & Engineering

MSc in Data Science
Project Report
2017/18

Neural Statistical Language model using Recurrent Neural Network Models and Word Vector Difference Representations

Francisco Javier Medel Medina

Supervisor: Dr Tillman Weyde

Acknowledgements

I would like to thank Dr Tillman Weyde for giving me the opportunity to take this project. His guidance as supervision are so much appreciated. I would like to express my deepest gratitude to Abinaya Sowriraghavan who supported me in each step in the realisation of this project for the companionship and the countless hours invested. Without her help, this simply could not have been possible.

I would like to thank my two sisters Denisse and Karla for their friendship and support.

Lastly, I would like to thank my Parents for their love, their valuable advice to overcome the obstacles that I found in this challenging road and the time on listening my thoughts and dilemmas, but especially my Mom Martha, she is my cornerstone in this travel that we call life.

Abstract

Word embedding representation shows a positive impact in the results generated by Natural Language Models. For instance, the use of Pretrained word vectors as Glove as embeddings helped to improve the task of predict words. The pretrained word vectors have as the primary characteristic the ability to representing semantic and syntactical relationships between words in a high dimensional space.

We propose to use the distance between word vectors as the difference vector in a language model and concatenate the difference vector with the vectors of each word vector representation. This allows us to feed additional structural information to different Neural Networks models (RNN, LSTM, GRU). We create three different model that manipulate this structural information differently to compare with baseline implementations to evaluate the perplexity chance in the word predictions.

We perform the hyper-parameter search that helps us to find the best model for each neural network. According to our results, the use of the differentiation vector in two of our models improves the perplexity of LSTM and GRU compared to the baseline model. Although the difference vectors lead to consistent improvements, these are not significant on the current, small number of experimental results.

Keywords: Difference Vectors, Word Embeddings, Word Prediction

Contents

List of Figures

List of Tables

1. Introduction and Objectives

Language models, i.e. Word Prediction Systems, are an important field of Natural Language Processing (NLP) research in academia, with many application areas such as Spell checking, Speech Recognition or Machine Translation. Generally, a Language model calculates the probability distribution of a token to appear in a given context. The aim is to predict the next word in a sequence, given a context word n-1, where n is the size of the Context Window that represents how many previous words are used to calculate the next word. The state-of-the-art of Language models take as a base Recurrent Neural Networks (RNNs) with its gated variants like Long Short-Term-Memory (LSTM) and more recently the Gated Recurrent Units (GRU) that are computationally less expensive than LSTMs and are showing comparable results. One of the aims of this project is to analyse the performance of these architectures by implementing different argument models from our domain.

The purpose of this project is to investigate whether the differences calculated between words in the context with a pre-trained word vector representation “Glove” can help improve the word prediction in Language models. Difference vectors are calculated by the element-wise subtraction between the vectors that represent the words in the context. Two papers that work as inspiration are GloVe: Global Vectors for Word Representation show that the use of embeddings improve the word representation (Jeffrey Pennington, Richard Socher 2014) and Language models with Pre-Trained (GloVe) (Makarenkov, 2016) where the use of GloVe indicated good results on related models using a word vector space on applications that include clustering and for some NLP applications, such as information retrieval, spell check or translation.

Another project that brings us more information on how argument models were implemented was the contribution done by (Taranpreet Singh 2017) that found how adding additional context to RNN models could help improve word prediction against traditional models, in this project they are performing Part-of-Speech (POS) tagging and augmented models that used embeddings by feed the Neural Network. Supporting this additional implementation. The thesis by Yu (2016), where models that had additional information in the input as POS tag embeddings show positive results in a new variance of Language models. The primary goal of this was the classification statements on the well-known Stanford Sentiment Treebank (Socher et al., 2013) dataset. These results open the possibilities to create a different model that can add additional information. This research area that has much attention from the community, the POS integration implemented in LSTM, and GRU structures showed improvements in the results.

The project revolves around two main ideas, Firstly, that we can create an argument model with additional structure in the input as the model with POS did by concatenating embeddings for the words of the context representation with their respectively embedding representation to feed the models. Secondly, models that had implemented word vector representation usefully using Glove have improved results in NLP different application. Taking the central concept of these two projects and help us to generate a new approach.

The method in this project is that we use the vector values from the words that were found in Glove, this matrix is going be used as Embedding input to the RNN models (RNN, LSTM & GRU). The main variation we add in this project is that we take the Embeddings values for each word from the context input and calculate the difference between the two vectors. Once we have this new vector, we use it in different implementations concatenating this vector with the vectors generated by the context words, e.g. in a way word1 + difference + word2 this create the input to feed the Neural Network. To the best of our knowledge, this idea is novel and has not been implemented anywhere else.

1.1 Research Question

Can the difference calculated between words from a pre-trained word vector representation "Glove", improve word prediction with Recurrent Neural Network structures on Language models?

Primary Objects:

- Develop an in-depth understanding of how the Neural Language model works in Deep Learning Structures.

- Develop a baseline and vector difference models that use Glove word vectors to achieve word prediction.
    - Implement a standard model that uses the Glove word vector as input.
    - Implement the vector difference model that uses the difference vectors as an extra feature.

- Deploy experiments with different configurations using a Deep Learning framework. Make the code modular for future projects.
    - Measure the model performance of different structures implementing a greedy search of parameters. Implement a comparable criterion of accuracy to evaluate the difference architectures RNN, LSTM, and GRU.

    - Can additional information such as the word difference vectors in the input of the RNN's model improve the perplexity of the models or help with other computational measures, such as the processing time.

    - Statistically evaluate the models to have a precise measure of use augmented model with Glove as embeddings and differentiation calculation.

1.2 Outcome and Beneficiaries

The purpose of this project is to contribute to the development of Neural Language models under the supervision of Dr Tillman Weyde to help to develop Research Centre of Machine Learning at City, University of London. We will benefit from this project as we aim to obtain skills and knowledge in one of the most growing areas: Natural Language Processing. We believe by contributing to this project; we will contribute to this field. The implementation of a model using a pre-trained word matrix can bring valuable results and further help improve models and develop new approaches for Postgraduate and PhD Students. The final milestone is to evaluate the accuracy of the text generation model in automatic translation application.

1.3 Tools and Software

Python 3.5 was the primary language used for developing this project. The main libraries to use were NumPy and Scikit-Learn. For a deep Learning the framework I used PyTorch (version 0.4.1). Data pre-processing techniques were implemented to prepare the data for further analysis.

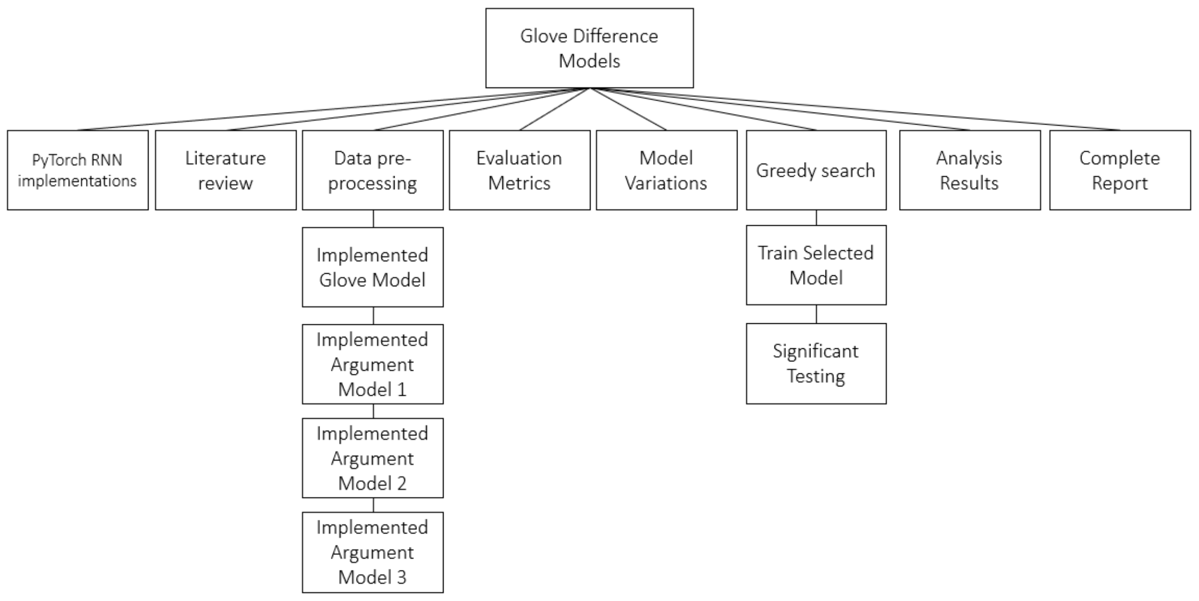
1.4 Experimentation Schedule

The first step is to start with the literature review to analyse the state of the art of Language model and how it can improve further aligned with this, we are looking to develop a project that can contribute to the community the Research Centre of Machine Learning at City, University of London. Then continue exploring different Language model Implementation on Deep Learning Architectures. Ones we are familiar with was already done and the literature was settled, we will proceed by creating small models to familiarise ourselves with the framework. Our premise was established, documented and evaluated the models and different approaches achieved.

1.5 Project proposal adjustments

The project proposal had an objective to implement predictions in a given context. The context was to include grammar rules as Background Knowledge (BK) one of the main ideas of the project was to use Logic Tensor Networks (LTN) Framework developed by the Research Centre of Machine Learning at City, University of London (Artur d'Avila Garcez and Luciano Serafini 2016). However, due to time restrictions, we opted to go for another research problem. Moreover, the fact that there are many complex and ambiguous concepts for a formal logic implementation. Plus, the need for implementing grammatical rules adds to the complexity of the Logic Tensor Network model. After thoroughly looking at the scope of this project, we decided to reduce the scope and focus on other vital problems that can bring valuable results on its findings. After a literature review, we realised that there are still areas to explore and one that took our attention was the word vectors from a pre-trained vector matrix Glove and, how the characteristic of distance feature can be implemented in a model to predict the next word in a Natural Language Process implementation.

1.6 High-Level work plan activities



2 Critical Context

In this section, we analyse the work done on Language models with pre-trained word vectors Glove. We explain how the embeddings representation helps in the Language model, then how this way of represented the words a vector an extender to a the pretrained word representation Glove, the critical characteristic of the Glove is the distance between the words vectors that represent a lexical context. The goal of the project is to develop Natural Language models using different ANN (Artificial Neural Networks) architectures. To complete the theoretical framework, we explain how the RNN (Recurrent Neural Networks) works and then compare the difference between a better variant to deal with a sequence of data the LSTM (Logistic Short-Term Memory) and GRU (Gated Recurrent Unit) models. One of the aims is to compare the performance of the argument models implemented in these structures. This project takes as a base the currently existing model implementations; our purpose is to validate if expanding their structures affect their performance positively.

2.1 Language models

A Language model is defined as a model that calculates the probability of predicting the next words in a given sequence of words. The aim of the model is learning the probability distribution from the context that preceded the predicted word (Kimet al, 2017). Some Language models use as a context a small sequence of words while other models use more extended sequence even complete sentences to calculate this probability. In our project, we are using the first approach the word level sequence. Where the probability distribution is defined as, E.g. $p({w_1}^T)$ in a sequence $w_1 = w_1, \dots w_t$ (Oualil et al., 2017)

$$p(w_1^T) = \prod_{t=1}^{T} p(w_t | w_1^{t-1})$$

The models that implement the N-Grams approach as smothers methods, where smoothing methods are a model that does not assign a probability to unseen data, The N-Grams showed very proficient results against their counterparts helping to improve speech recognition implementations at that time (Chen and Goodman, 1998). In this model, each word is represented by a token in large vocabulary size, on this started implementation was common that the results show a poor probability estimation. This was due to what Bellman (1957) described as Curse of Dimensionality; It explains that is difficult to analyse large datasets starting from the point that by all the possible variables in n-dimensions, the space between the data values grows exponentially.

The curse of dimensionally in Language model is due to the high number of words in the language and the different combinations of sequences that can come before a predicted word. Also, the language itself has an innumerable amount of terms that can bring ambiguities. One of the most common scenarios in Language models is that the context given in the training set does not necessarily occur in the same order in the testing phase even when the aim is to predict the same word in both samples.

Trying to deal with the Curse of dimensionality with the usage of N-Grams was used a concatenation of a short number of sentences as the context in the training set (Bengio, 2003). A different technique was to associate each word with its corresponding word feature vector and use this as an input for the Neural Network. This new implementation was formatting word embeddings that brings real value in a word vector space; it helps to implement a more extended context.

The Neural Networks implementations show better results than the classical methods. The resurgence started in Neural Networks used the Feed-Forward Neural Network (FNN) structure using a fixed length-context on Neural Net Language models (Bengio, 2003) this paper settled the base for a future Statistical Language models that opened the door to a more sophisticated Recurrent Neural Networks (RNN) structures.

2. 2 N-Grams

Our base model is the N-Gram model one of the most widespread paradigms in the Natural Language; the N-grams is a sequence of the n item in a string what follows the $(n-1)$ Markov assumption (Chelba, 2017)

$$P(w_i|w_{i-1}, w_{i-2},\ldots,w_{i-n+1}) \tag{1}$$

W_i is the word in a sequence (N-Grams PyTorch, 2018)

The main two advantages of the N-grams models are scalability and simplicity, with a bigger n we can give more context, it makes easy to scale the type of experiments. N-grams fits perfectly to our needs; we need at least two vectors to calculate the differentiation that allows creating a new vector value that will use as an additional feature to feed the ANN structures (RNN, LSTM, GRU). We plan to implement trigrams for this project, which means two words as context and one word to be predicted.

A different implementation that used nested features as back-off n-grams in Language models has shown excellent results compared to the parameter stored in the model against the number of features (Pelemans et al. 2016).

One of the limitations of the N-grams is that the words that are not present in the vocabulary out-of-vocabulary (OOV) words, these words are not fed to the Network during the training phase. One of the ways to avoid this obstacle is to define a fixed size vocabulary. In this way, these words are ignored as denoted by the label <unk>. As a result, the probabilities generated of each word in the vocabulary are smoothed even when they are not presented in the training phase.

2.3 Recurrent Neural Network (RNN)

For this project, we are focussing on the Recurrent Neural Networks (RNNs) implementations, where the main difference with the Feed-forward is that the RNN incorporate a temporal loop that allows the model to retain information from the previous sequence of data. Bring the capacity to deal better with a series of data. The term "recurrent" mains that the task is performed over the same instance of the sequence, such each output is independent of the previous instance (Elman, 1990).

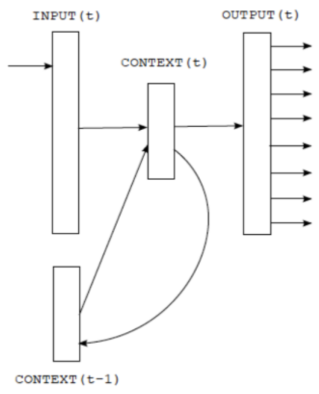


Figure 1: Architecture of an RNN (Mikolov et al., 2010)

(Mikolov et al., 2010) Implemented the first models applied to Language models using Recurrent Neural Networks (RNN). The results showed that this model overperforms state of the art at that time back-off Language models, indicating a lower perplexity with a smaller amount of data given in the training phase, one aspect to considerate is that the complexity regarding computational cost was higher.

This implementation uses the maybe the simpler version of the RNN or also called Elman network (Elman, 1990) follows a network structure with an input layer x, hidden layer h (context layer) and output layer y. The input vector of the network in time t is $x(t)$, the output vector is denoted as $y(t)$, and $h(t)$ is state of the network hidden layer vector, σh and σy represent the activations functions of the layers. W, U and b contain the matrix parameters.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \quad (2)$$

$$y_t = \sigma_y(W_y h_t + b_y) \quad (3)$$

Each word takes its corresponding value from the word vector representation, it is used to feed the network as the Input vector $x(t)$, in the output layer we got the predicted values, the in-context layer h is continuously updated at time $t-1$ using the context of the last word (Oualil et al., 2017)

We can denote the basic formula of the RNN applied to the Language models (Mikolov et al., 2010). This model can process an input sequences context vector in the hidden state h_t: the memory of the Network, in each time step, h_t is the function for the previously hidden state h_{t-1} in the current vector input $x(t)$, where the Network gets in the time t and the previous h_{t-1}.

$$h_t = g(W_{x_t} + U_{h_{t-1}} + b) \quad (4)$$

W and U are weight matrixes, b is the parameters of the function x_t is n-dimension vector representation for the word in the context in the time t, g works an activation function.

This equation shows the usage of the hidden state to predict the next word, and how it depends on the probabilities given by the previous words, this information is stored in the hidden state (Oualil et al., 2017)

$$p(w_1^T) \approx \prod_{t=1}^{T} p(w_t|w_{t-1}, h_{t-1}) = \prod_{t=1}^{T} p(w_t|h_t)$$

2.3.1 The Long-term dependency problem

Thanks to its structures the RNN's can retain information in the hidden state but, the main limitation for the standard RNN is that the difficulty to solve problems relate to high temporal dependencies.

In the research done by (Bengio, Y 1993) explain that the gradient loss function decay over the iteration in an exponential way happening in long sequences this known as *vanishing gradient problem*. This next issue occurs when the error of the loss function is calculated and back-propagated into the layers of the Network to update the weights, the gradients tend to be smaller and for the early layers, and as a result this layer learn to slow compared to the layer that is closet the output layer (Anish Singh, 2017)

Some of the approaches solved this problem could be to set the initial weight of the Network or use prior knowledge, which seemed to have dealt with such issues (Bengio, Y 1994). Another option is to avoid the use of a Sigmoid or Tanh function that contributes to this problem instead of Rectified Linear Unit (ReLU). This function allows the Network to continue learning even when the inputs are very small or negative. We can understand as a binary solution, where when a value exceeds a threshold it can be taken to perform the learning step of otherwise the values does not overcome the threshold leave the neuron without the learning step, even when it does not resolve to complete the problem it helps to face in a smoother way (Rohan Kapur, 2016).

2.4 Long Short-Term Memory (LSTM)

An RNN variant that was able to manage the long sequences problem is known as Long Short-Term Memory developed by (Hochreiter, Schmidhuber, 1997) had made a significant contribution to eradicating a variety of issues. Later on, the model was improved (Gers F.A., Schmidhuber, 1999) with a new characteristic of "forget gates" Input gate, out the gate, forget gate and cell state. It remembers values in t intervals; the other three gates regulate the flow of information across the neuron, which is a standard LSTM version. It is the architecture that we are going to use in our experiments.

The key behind the LSTM is the cell state or memory cell. The "memory cell" helps to retain information for an extended period. The gate into the cells helps to control when the information enters to the memory, for how long keep the information and when it is out to release the memory. This architecture helps to learn Long-term dependencies (Chung et al., 2014) more efficiently.

The LSTM was developed with the idea of dealing with the vanishing gradient descent problem by applying this gating technique. The RNN and the LSTM's have similar structures, but the standard RNN contains just one layer that implements the Tanh function. The LSTM has four layers performing in two layers the Than function.

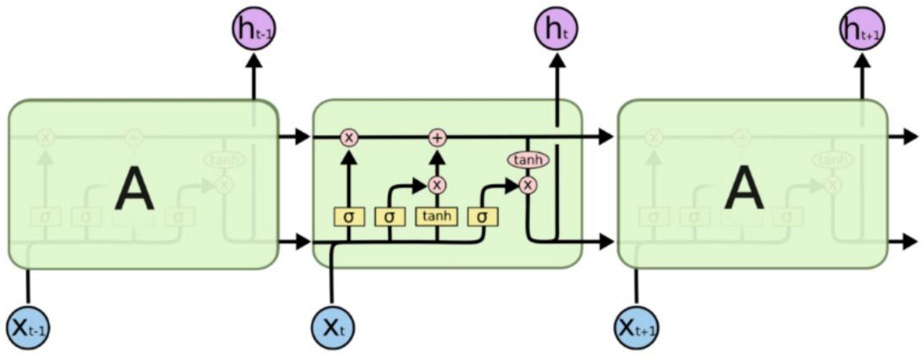


Figure 2: Architecture of an LSTM (Olah, 2015)

The LSTM uses the equation in (7), Internally each gate uses a sigmoid function, where a value of one to the neuron means "learn everything" and a value of zero means "learn nothing". In the RNN structure, the Hidden layer plays the role of the memory cell, c_t.

To get the memory state c_t the hidden stated is added with the weights-sum denoted by $\odot$ function with the early memory state indicate by c_{t-1} state. The equation 6 and 7 uses a $Tanh$ denoted by g as not linearity activation function to process the gating signals (Dey and Salem, 2017).

$$c_t = f_t \odot c_{t-1} + i_t \odot c_t \tag{5}$$
$$c_t = g(W_c x_t + U_c h_{t-1} + b_c) \tag{6}$$
$$h_t = o_t \odot g(c_t) \tag{7}$$

How LSTM manage the iteration is one of the critical aspects of the model. The equation 8, 9, 10 denotes the equations of the gates. In a sequence time t for input (i_t), output (o_t) and forget (f_t), the difference with the equation 2 is that the $Tanh$ function is changed for σ a logistic function with the limits between [0,1].

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{8}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{9}$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{10}$$

All this internal calculation into the model can be interpreted as a black box, where given an input hidden stated and the previous one it computes the next hidden state in a sequence of data.
The functionality of the memory cell can be extrapolated in a Language model as a subject that takes care of the state of the verb in a sentence. The form of the subject will denote the conjugation of the verb if the subject is singular or plural.

LSTM's have shown to overcome the RNN algorithms, mainly due to the higher computation cost in the internal layers by managing the gating technique when compares to a simple RNN. As a result, the perplexity in Language models has reduced scientifically applying this architecture (Zaremba et al., 2014).

LSTM's have helped to resolve ones of the most challenging problem in computer science, its application in Speech Recognition or Script Translation, as an example applied in the industry, LTMS helps to boost the Voice Transcript by Google, a speech recognition implementation that uses the feature of recurrent connection that makes it able to "remember" the sequence of words in a long sequence, this new implementation overperform the previous version of the same Speech Recognition system launched two of year before (Beaufais, F., 2015).

2.5 Gated Recurrent Units (GRU)

Gated Recurrent Units (GRU) another RNN variant, GRU has a similar architecture that the LSTM but simplified. The memories cells are not separate from one another, and they count with few gates to controls the "retain" and "forget" memory iterations. This architecture has just two gates instead of three (Cho et al., 2014). This simplified structure takes less effort to train the models (Kaiser, 2015). Even when this model is not as complicated as counterpart LSTM, the results have shown to be very similar to the implementation in phonetic music (Chung et al., 2014). This characteristic makes GRU a natural option to compare with LSTM in our project.

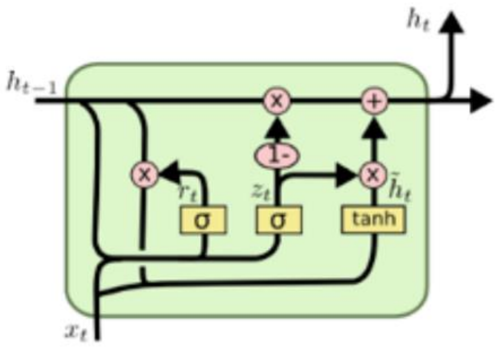


Figure 3: GRU Cell (Olah, 2015)

As we can see in the figure 3 the internal architecture of GRU is quite similar to LSTM, but the main difference is that GRU does not have an output gate whereas LSTM does. GRU has two gates are z_t and r_t update and reset respectively.

In this model, the update gate controls how much information is retained per iteration, and the reset gate controls the combination of the previous memory state with the new input. GRU can work as a simple RNN when the update gets is 0, and the reset gate is equal to 1.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h_t \tag{11}$$
$$h_t = g(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h \tag{12}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{13}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{14}$$

The functions for GRU are denoted in the equations (11) and (12), similarly to LSTM g use a $than$ function no linearity function. The equations for the gates are the (13) and (14) respectably.

There is some difference between GRU and LSTM, as GRU does not have internal memory (c_t) and it does not have out gate too. On this model, the two gates form LSTM forgot, and input is replaced for an update gate. Moreover, because the GRU does not have an output gate, this model does not apply second nonlinearity to process the output of the neuron.

Now that we are familiar with the internal structure of the main RNNs and their difference, we can think about which of these models are better to Natural Language process, but the current results in few researches show that this question is not as easy to understand, the performance very similar to result in the same problem. Tuning hyperparameters can be more important than just choosing the best architecture.

Both architectures are going to be implemented in this project; the main idea is to apply additional structural information to the input of the models and compare the performance against models that do not incorporate this additional feature.

2.6 Incorporating the Embeddings Layer

In the area of Machine Learning, many variations can improve some problems, such as adding additional features to the model. We can apply the same principles to Natural Language Processing (NLP).

(Press, O. 2016) Explains how using Embeddings help to improve Language model, using a topmost weight matrix generated from the output embeddings can be used a valid word embeddings, he proposes of using updated rules on tied input embeddings cab evolve in a better way to output Embeddings resulting in a lower perplexity against a model that does not implement tied inputs.

The embeddings can store some semantical information; it proves to be useful for a model that can improve the finding of similarities in text and can be replicated for similar tasks. Word embeddings have a significant contribution to the state-of-the-art results for NLP (J. Weston, 2011), (R. Socher, 2011), (P. D. Turney, 2010), (E. Cambria, 2017).

2.7 Embedding representation

The advantage that Embeddings have given to the Language model cannot be underestimated. Depending on the type of embeddings implemented, either similarity relationship or semantic representation, they have shown improvements in different areas helping to uncover patterns into the data. Semantic information has helped improve the results on some areas as parses trees or part of speech tags (Dense Embeddings PyTorch, 2018), In the field of NLP, they have boosted the results in Sentimental Analysis (Socher 2013) and Syntactic parsing (Socher 2013).

The word embeddings give a numerical representation of real numbers in a vector to each word in the vocabulary (Word Embeddings PyTorch, 2018). The embeddings are stored in a matrix of V x D where V is the vocabulary length and D represent the vector dimension. We need to create an index for each word in the vocabulary; it would work as a lookup-table to retrieve the word vector value that would be used as an input to feed the Neural Net. Word embeddings help improve the calculation of word similarities thanks to the characteristic of the low-dimension matrix (Levy, O, 2014).

2.8 Glove pretrained word vector definition

Glove a word representation use an unsupervised learning algorithm to obtain these vectors (Jeffrey, Richard, 2014). In a corpus, a correspondence matrix is calculated to get the co-occurrence of statistical information by word to word for the log-regression model, in this way the word similarity is the main feature for the model. The Euclidean distance can give us a good representation or the measure between two words vectors, where the linguistics or semantics is correctly represented. This is the concept of the Nearest Neighbours, for example, the closest words in Glove for *lion* and *England* in figure 4.

Lion	
Euclidean Distance	Closest Word
2.3751655	dragon
2.6790958	beast
3.0598903	unicorn
3.1958214	elephant
3.4052489	cat
3.4462893	bear
3.5480326	golden
3.5883568	peacock
3.6878726	rabbit
3.6992118	mermaid

England	
Euclidean Distance	Closest Word
2.3751655	scotland
2.6790958	wales
3.0598903	ireland
3.1958214	newcastle
3.4052489	australia
3.4462893	manchester
3.5480326	zealand
3.5883568	indies
3.6878726	scottish
3.6992118	leeds

Figure 4: Closet words for lion and England using Euclidean distance

The distance between two words can result in a simple scalar value; this can be a problem because of the relationship that two words can contain an intricate relationship. For example, the words man a woman are commonly associate as be opposite but, in some context, they can play the same role as a subject acting in the sentence, where this main of opposites does not have any meaning.

In order to show a quantitative value to distinguish two words is necessary to give more than a straightforward number to have a correct representation. Glove uses the juxtaposition vector values in the vector space of two words to calculate the vector differences. This pattern is exhibited in figure 5.

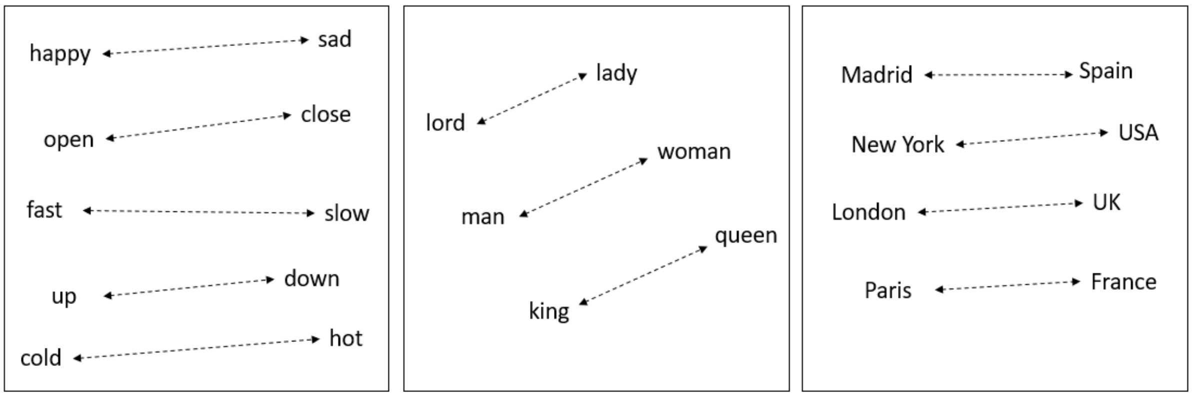
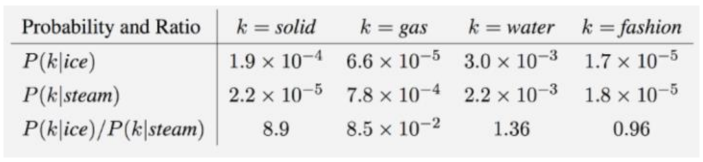


Figure 5: The Concept *of Distance between words*

The main idea of this model is to calculate the probabilities of the co-currencies by simple observation by word-word, it has the potential to getting some meaning of the target. For example, the word ice and the number of times this word is followed by other words of the vocabulary in the same string. These probabilities were calculated in a source of 6 billion tokens, figure 6.



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 6: word-word co-occurrence probabilities

As a result, we get a co-occurrence table with the ratio of probabilities. We can see whom the word ice co-occurs more often with the word solid that the word gas and the word stream co-occurs more often with the word gas that the word solid. Also, we can see the unrelated probability of the of the word water with fashion. In this example, Checking the probability and Ratio results in the values greater than one correlate well with the word ice and the values lower than one correlate well with the word steam (Jeffrey, 2014).

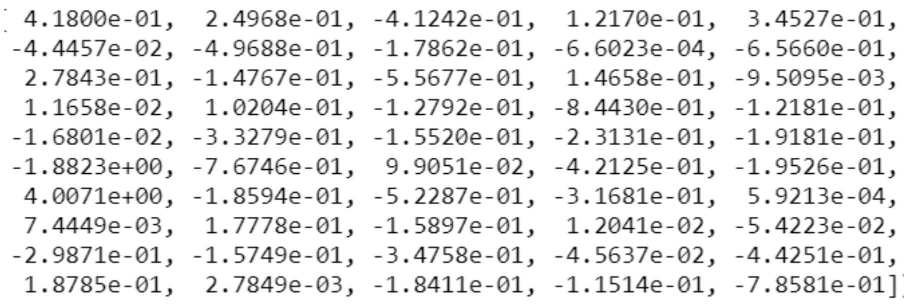
The aim for Glove is to generate word vectors where their dot product logarithms represent the probabilities of the word co-occurrences.

2.9 Word Embeddings as pretrained word vector

One of the most popular word vector representation “word2vec” (Church 2017) for word embeddings helped to improve two of the most popular architectures, Skip-gram model and Bag of Word (CBOW) (Mikolov et al., 2013). In other, the hand Yu (2016) tried to implement word2vec in POS tag, but this new approach did not show improvements in the results in otherwise the standard implementation of generating a random normal distributed value to represent the word as embeddings are more appropriate to solve this kind of problems.

The work was done by (Makarenkov, 2016) as a cornerstone for our project. This work shows improvements in word prediction results for Language models. This implementation outperforms other popular Embeddings representation as word2vector on a specific task (Makarenkov, 2016). This work implementation developed an RNN structure that was shown to beat most of the traditional Language models. On this project, a GRU Neural Network was used to memorise the sequence to given words. The model implements Glove in a dimension size equal to 300 to 50 from 42 million tokens in a 1.9 million vocabulary size.

For each word in the vocabulary, the corresponding value is looked in Glove if the word is not present in the pre-trained vectors, we create a new random vector with the same size dimension. It creates a vector matrix for all the text corpus. The vector-matrix will then be used to load the pre-trained vector values instead of training from scratch into the model.



```
 4.1800e-01,  2.4968e-01, -4.1242e-01,  1.2170e-01,  3.4527e-01,
-4.4457e-02, -4.9688e-01, -1.7862e-01, -6.6023e-04, -6.5660e-01,
 2.7843e-01, -1.4767e-01, -5.5677e-01,  1.4658e-01, -9.5095e-03,
 1.1658e-02,  1.0204e-01, -1.2792e-01, -8.4430e-01, -1.2181e-01,
-1.6801e-02, -3.3279e-01, -1.5520e-01, -2.3131e-01, -1.9181e-01,
-1.8823e+00, -7.6746e-01,  9.9051e-02, -4.2125e-01, -1.9526e-01,
 4.0071e+00, -1.8594e-01, -5.2287e-01, -3.1681e-01,  5.9213e-04,
 7.4449e-03,  1.7778e-01, -1.5897e-01,  1.2041e-02, -5.4223e-02,
-2.9871e-01, -1.5749e-01, -3.4758e-01, -4.5637e-02, -4.4251e-01,
 1.8785e-01,  2.7849e-03, -1.8411e-01, -1.1514e-01, -7.8581e-01]
```

Figure 7: Vector values of the word ‘the’ for 50 dimensions

As a result, this model shows the small error to predict the next word against the embeddings that was generated randomly. One of the main limitations of this implementation was the context size, in the Language models (LM). Other areas of research are open as a dynamic context size with the use of other packages as DyNet (Neubig, 2017). This can give the flexibility of dynamic sizing to the LM and the ability to perform dynamic computational and sharing the learned weight across the different variables in the training phase.

2.10 Conclusion Critical Context

From the literature, it is clear that there is much room to implement new approaches in NLP applied to Language model.

The use of LSTM helps to develop some state-of-the-art models using different approaches. There are other methods as the embeddings that contain syntactical information, are used to convert the word in a vector space that can be used to find a similarity of words. Glove give brand-new approach with a word vector representation helping to integrate additional information as the distance between two words vectors that can bring a lexical context to create augmented models that can take this characteristic.

The main idea is to investigate whether RNN's structures can take advantage of Word vector representations as an additional feature to feed their structures. The following sections will elaborate on the methods used and the results obtained.

3 Methods

In this section, we introduce the methods that we follow to achieve the goals that we explained in the introduction. Firstly, we perform data analysis on the dataset. We explain the hyperparameters that make up the structure of the RNN's models, why we chose to implement the Grid search how it can help us to find a model that generates lower perplexity. A description of the hyperparameters and how we slip in two group to perform the different experiments. Moreover, for the last step, we explain a method to calculate how significant are the model's results.

3.1 Tools and Software

This project was developed in the Programming Language Python in its version 3.5.2. The main libraries used to explain the Python characteristics are listed below:

- NumPy: for arrays and matrix manipulation.
- PyTorch: Deep Learning framework.
- Pandas: Data Analysis and Data Manipulation.
- Time: For processing time measures.
- Math: Perform mathematic operations.
- Matplotlib: Data visualization and Data Representation.
- NLTK: Data Pre-processing.
- String: Data Pre-processing and String formatting.

The project was done in two parts, the developing phase was done in a Windows laptop, with a tiny part of the full dataset, with 35,000 samples for training, 5,000 samples for testing and a vocabulary size of 1,000 words. As well as the first steps, tutorials on PyTorch was taken to improvise on programming knowledge and to get expertise in the Deep Learning framework.

Once we have a script that runs well in this developing environment, we migrated to the City servers to start the training phase with the dataset selected for this project the Penn Treebank with 169,090 samples for training, 74,908 samples for testing and a vocabulary of 10,00 words, with this dataset with start to recollect measures and record the results of the different experiment. In the table below, we can see the server specifications.

Server Name	GPU Model	GPU's	RAM GPU	CPU Model	Cores	RAM CPU
America	GeForce GTX 1080	2	11	AMD Ryzen 7 1700	16	32

3.2 Dataset, Data Prepossessing and Data Analysis

The selected dataset for this Natural Language project is the famous Penn Treebank dataset (Marcus et al., 1993). This dataset is spliced in training, testing and validation sets. Each set is composed of sentences. Starting with the prepossessing phase, the first step was to convert all the sentences to lowercase to define what the word "monday" has the same representation as "Monday" in the dictionary. Secondly, we mark the end of a sentence with the label <eos> to eliminate the "\n" character, this mark helps us avoid the problem of mixing sentences denoting the end of each one.

One we have the train set in lowercase, and with the label <eos>, the next step was to apply a tokenisation in the set to get vocabulary of the corpus. We are getting a vocabulary of 10,000 different words from the training set. It is important to mention that Test and Validation sets cannot contain a word that does not exist in this vocabulary, for this version the Penn Treebank is limited to a universe of 10,000 unique words. The models cannot generate a probability for words that have not seen before in the training phase and are not a member of this universe of 10,000 words.

['aer', 'banknote', 'berlitz', 'calloway', 'centrust', 'cluett', 'fromstein', 'gitano', 'guterman', 'hydro-quebec', 'ipo', 'kia', 'memotec', 'mlx', 'nahb', 'punts', 'rake', 'regatta', 'rubens', 'sim', 'snack-food', 'ssangyong', 'swapo', 'wachter', '<eos>'] ['pierre', '<unk>', 'N', 'years', 'old', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'director', 'nov.', 'N', '<eos>'] ['mr.', '<unk>', 'is', 'chairman', 'of', '<unk>', 'n.v.', 'the', 'dutch', 'publishing', 'group', '<eos>'] ['rudolph', '<unk>', 'N', 'years', 'old', 'and', 'former', 'chairman', 'of', 'consolidated', 'gold', 'fields', 'plc', 'was', 'named', 'a', 'nonexecutive', 'director', 'of', 'this', 'british', 'industrial', 'conglomerate', '<eos>']

Penn Treebank Tokenized sample

In table 1 we can see a summary of the total number of tokens present in the Peen Treebank dataset, a token as is a word in the dataset. These tokens are going to be used to create the trigrams to feed the models. Each one of the trigrams will work as the context for the predicted words.

Tokens			
Dataset	%	Tokens	Total Number of Tokens
Train	85	929589	1085779
Test	7	73760	
Validation	8	82430	

Table 1: Number of tokens in the Penn Bank Treebank Dataset

3.3 Codebase

The starting point to develop the model was taken by PyTorch Documentation; the code follows the structure explained in the PyTorch Tutorials for Language modelling implementation. The code was designed from scratch; this brings us the freedom to develop out tasks such as pre-processing, data-analysis, model-creation, perform a greedy search, and graphs to visualise the outputs.

3.4 Evaluation Metric – Perplexity

In a Natural language Processing, a common way to evaluate the models is the use of Perplexity, which measures the probability distribution of a word in a sentence. The goal of the models is to have a lower perplexity that minds that the model is good at generalising a word sample in vocabulary.

To calculate perplexity, we need to compute the Cross-Entropy that is used to measure the performance between the two distributions. This measure averages the number of bits required to identify an item in a set.

$$H = 2^{-\sum_x p(x) log_2 p(x)}$$

The PyTorch Cross-Entropy Loos function calculates the losses averaged over the sample mini-batch size on the model prediction for a word set. With this Cross entropy calculated we apply a simple notation, exponential on base 2 to the Cross-Entropy to get the perplexity.

$$PP = 2^{H(p)}$$

In PyTorch Cross-Entropy Loss combines in one function, the log-SoftMax and NLL-loss functions that also is commonly used to calculate the perplexity, Applying Cross-Entropy Loss on the model prediction eliminated the need of adding an extra layer to the model (Pytorch.org, 2018).

3.5 Glove Baseline Language model

The base model is going to be implemented in the Three ANN cells type, RNN, LSTM, and GRU with the same setup for research purpose; we want to compare the results showed by the three cells. The internal structure has been covered extensively in section 2 critical context to understand better the inner working; please refer to this section.

Our base model is an N-Gram in the order of n = 3 also knows as Trigrams. Each model computes only one Trigram per iteration and calculates the probabilities to predict the next word in the sequence. At the start, the cell memories are set up by PyTorch, and it is suddenly updated while every Trigram is processed through the network. A mini-batch was set up as 16; each batch size represents one-time iteration. The batch size was select on base to work done by (Li, M et al., 2014).

The forward pass denotes the calculation of the input values to the output data, where it flows from the first layer the input of the network, through all the hidden units of the hidden layers up to the last layer. The backward perform updates of the weights of the network through the gradient descent optimisation algorithm. Both forward and backward passes are completed on one iteration during the training phase.

In each Mini-batch, the sequence of trigrams is presented to the model as a tuple of numbers. Following the standard implementation in PyTorch (N-Grams PyTorch, 2018), the model has an embeddings layer that converts these numbers to a vector that are used to feed the Neural Network. The weights are initialised between [-1 and 1] at the moment that the model is created.

Each model has two hidden layers to take advantage of Deep Learning implementation by adding a depth dimension to the RNN. Add more than two layers can increase the time to the train the model can also contribute with the problem of vanish gradient problem. A regularisation technique has been used the dropout to prevent overfitting on the train set and generalise better model on unseen data (Gal & Ghahramani, 2015).

Learning Rate is set up with a value of 0.1 for the experiments, and the value stays constant during the training phase, a value of 0.1 shows good results to create a reliable model (Lipton, Z.C., 2015). As an optimisation function, we use the stochastic gradient descent (Ruder, 2016) that updates the parameter after each iteration.

The criterion measures the gradient against a loss function to perform the minimisation in error. For the loss function, we use the Loss Cross Entropy Function by PyTorch; this function returns an average over all the samples sent in the mini-batch. The goal is to minimise Cross-Entropy during the training phase; it will be used to calculate the Perplexity of the model as a final measure.

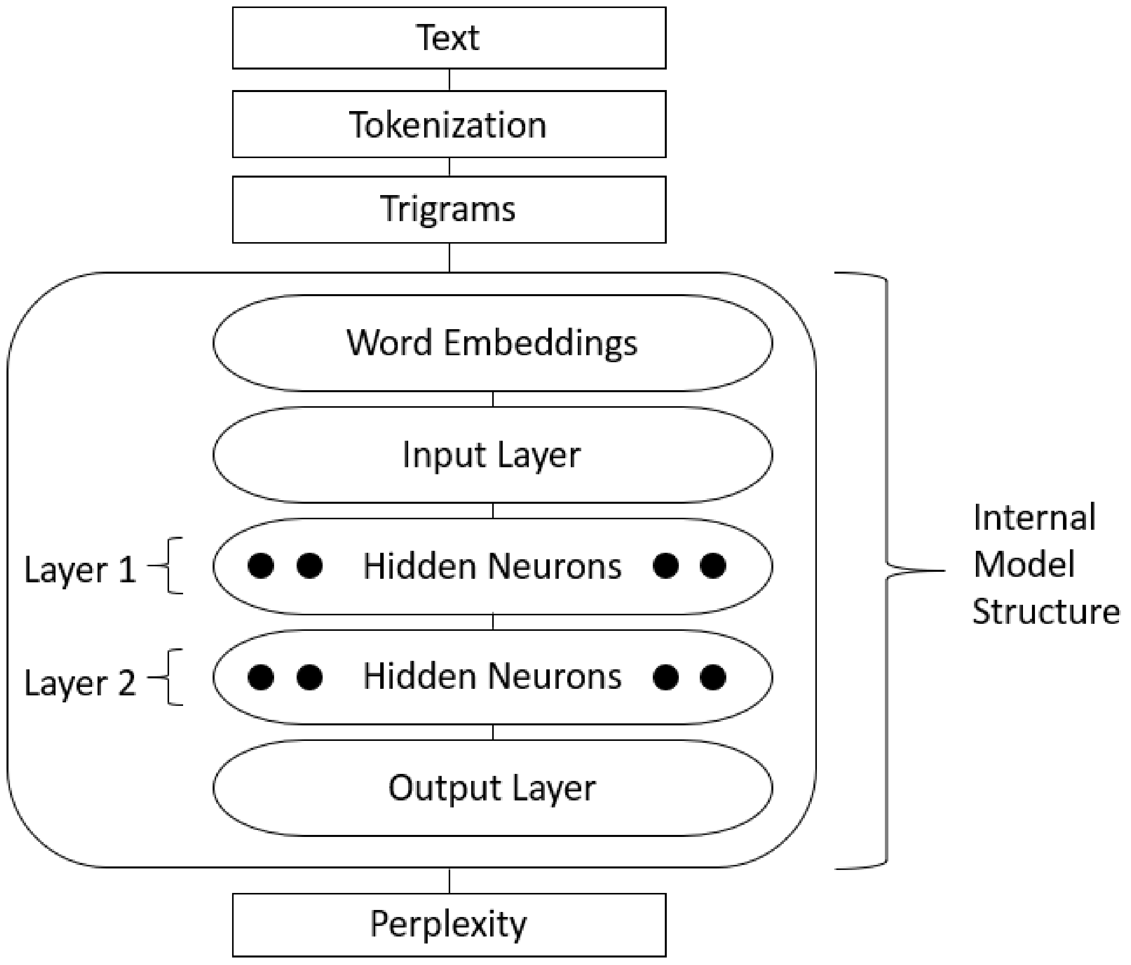


Figure 8: Standard Architecture of Baseline Model, each layer has implemented RNN, LSTM or GRU cells. The model processes the Word embeddings layer inside the model

3.5.1 Other approaches for the Embeddings Layer

The standard implementation to solve a Natural Language model in PyTorch (N-Grams PyTorch, 2018), Manage the creation of the embeddings internally into the model, in the training stage, a trigram is presented to the model, the trigram is built by an index representation of the words in the form (context, target) where context is a tuple of two scalar values, these values are used to search in a lookup table to retrieve the vector to feed into the Neural Network.

We want to try other approaches and modify this architecture to leave the job of creating the model to the RNN, taking out this Layer from the internal structure of the model. The difference is that we are presenting the vector representation directly to the network in the input layer, no a tuple that contains the index of the two-words that represent the context that the model has to use internally in the embedding layer to get the word vector representation.

We have a hypothesis that with this modification of taking the job of the embedding layer out of the model can improve the model performance. We are going the same job as the embedding layer in custom function, taking the tuple that contain the two-words indexes from the context part of the trigrams and find the corresponding vector representation, the critical part here is that we are doing this process just one time before running the model as pre-processing step no as a training step.

We expect that with this modification on the model structure, we can reduce the computational cost of the training phase, simplifying the internal structure of the model on removing one layer. We ran a couple of experiments comparing these two approaches, in section 4.1 where we can find the results of this experiments.

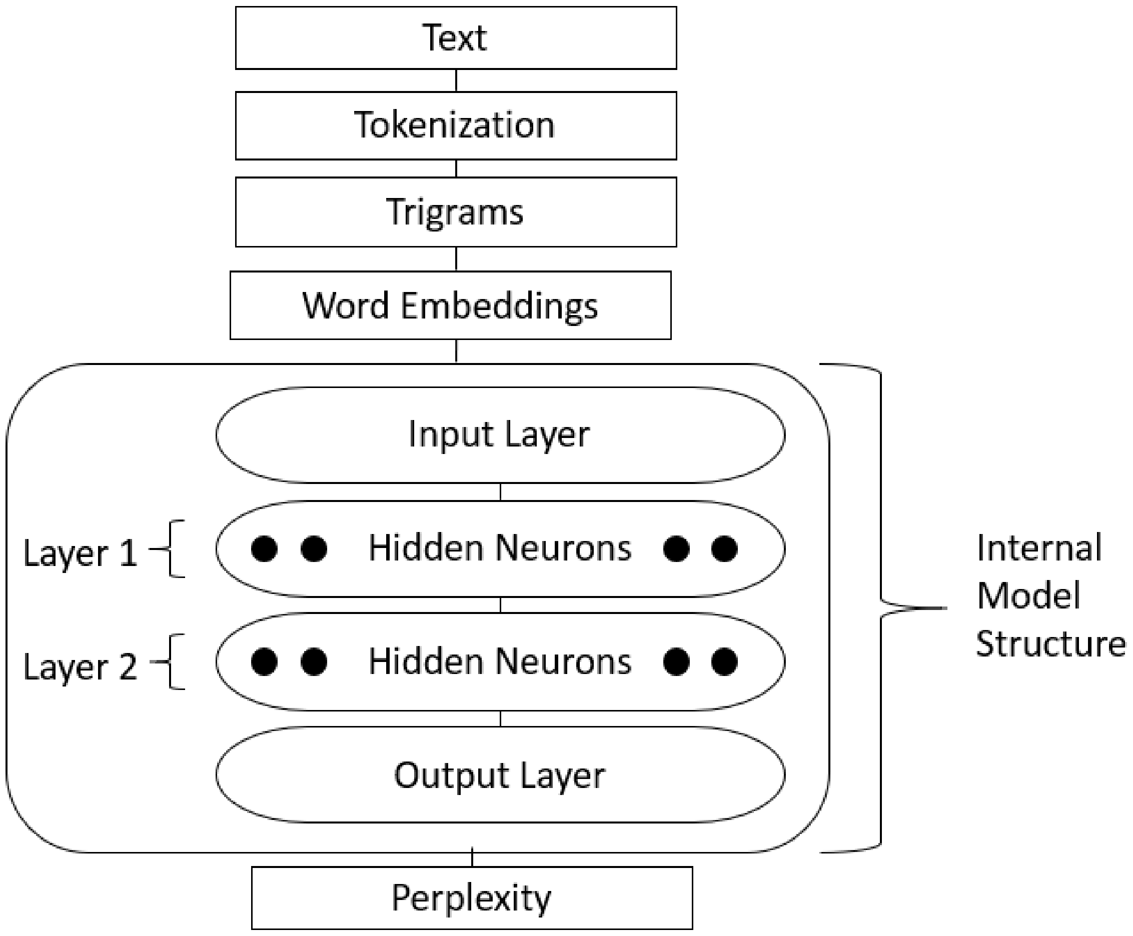


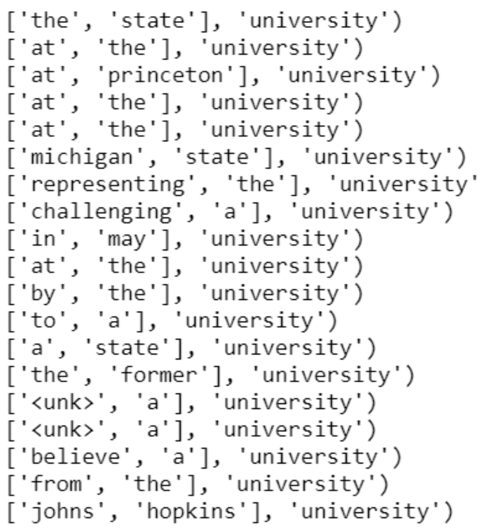
Figure 9: Standard Architecture of Baseline Model, each layer has implemented RNN, LSTM or GRU cells. The pre-processing phase processes the Word embeddings layer outside the model.

3.5 Beyond the Pre-processing

The Natural Language Toolkit (NLTK) library specialised in Natural Language Processing application helped us the create the tokenisation, cleans and adds the necessary label to the data.

Once we have the tokenisation, we need to create the trigrams to follow the structure necessary to build an N-Gram in order 3. The trigrams are composed of two components, the Context and the Targe. In the context side, we have two words to predict the target. These words are converted to indexes to reach into a lookup table.

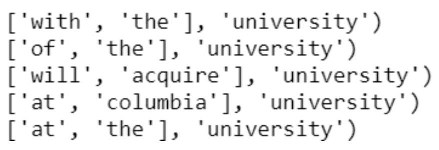
To create these trigrams, we took into consideration the label <eos>, it references the last token in the same sentence. The following sample shows how many trigrams are composed by a pair of context and target. Each trigram is a sample in the dataset. For example, the word "university" produced the bellow combinations across the dataset figure 10.



```
['the', 'state'], 'university')
['at', 'the'], 'university')
['at', 'princeton'], 'university')
['at', 'the'], 'university')
['at', 'the'], 'university')
['michigan', 'state'], 'university')
['representing', 'the'], 'university'
['challenging', 'a'], 'university')
['in', 'may'], 'university')
['at', 'the'], 'university')
['by', 'the'], 'university')
['to', 'a'], 'university')
['a', 'state'], 'university')
['the', 'former'], 'university')
['<unk>', 'a'], 'university')
['<unk>', 'a'], 'university')
['believe', 'a'], 'university')
['from', 'the'], 'university')
['johns', 'hopkins'], 'university')
```

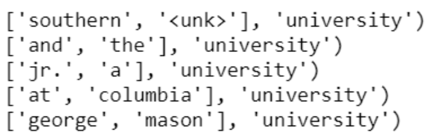
Figure 10: Trigrams for the word "university" in the Train set

We create trigrams for all the sets in the data source, Penn Treebank has three sets Train, Test and Validation. One aspect to consider in the Language model is that the Test and Validation could contain trigrams representing a word that no necessary presented the same order or even with some context to the model in the training phase. It increases the complexity of the Language model and is one of the challenges in this kind of model, trigrams generated by the word 'university' Figure 11.



```
['with', 'the'], 'university')
['of', 'the'], 'university')
['will', 'acquire'], 'university')
['at', 'columbia'], 'university')
['at', 'the'], 'university')
```

Test set



```
['southern', '<unk>'], 'university')
['and', 'the'], 'university')
['jr.', 'a'], 'university')
['at', 'columbia'], 'university')
['george', 'mason'], 'university')
```

Validation set

Figure 11: Trigrams for the word "university" in the Test and Validations sets.

All the words in the vocabulary are mapped to indexes that allow us to search the values in the lookup table. In our dataset, there are 10,000 different words. In figure 12 illustrate a Trigram for the word 'university' mapped to the indexes that represent each word in the vocabulary.

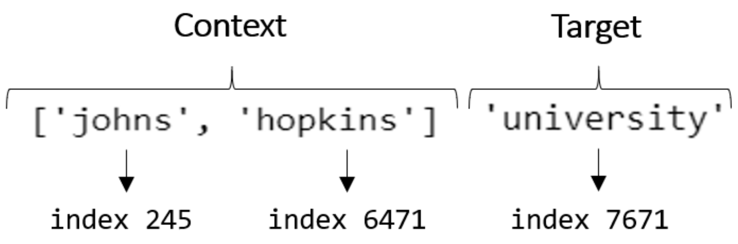
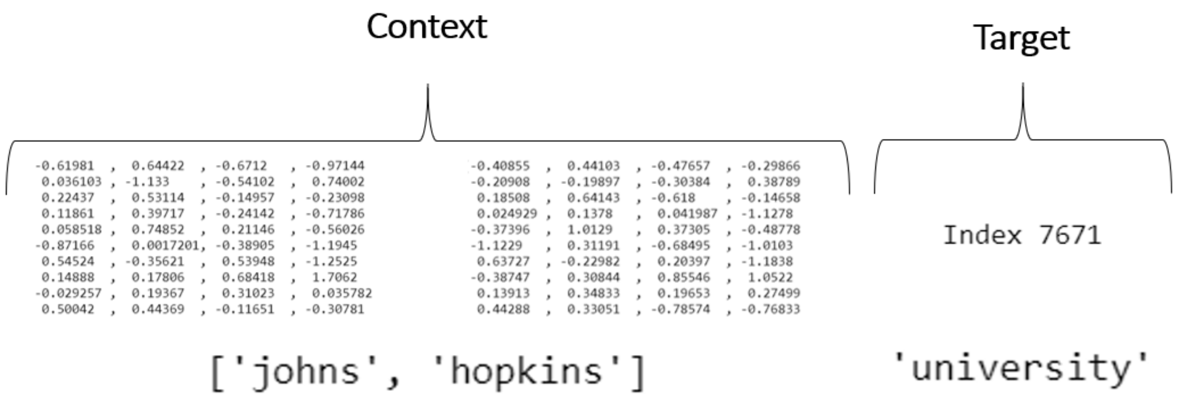


Figure 12: Trigram word mapped to indexes

After mapping the trigrams to indexes, we have a trigram make up for scalar values. We use these indexes to look their vector values in Matrix that contain all the vector for each word in the vocabulary. These indexes can be converted to Embeddings vectors that will be used to feed the Neural network Figure 13.



```
-0.61981  ,  0.64422  , -0.6712   , -0.97144
 0.036103 , -1.133    , -0.54102  ,  0.74002
 0.22437  ,  0.53114  , -0.14957  , -0.23098
 0.11861  ,  0.39717  , -0.24142  , -0.71786
 0.058518 ,  0.74852  ,  0.21146  , -0.56026
-0.87166  ,  0.0017201, -0.38905  , -1.1945
 0.54524  , -0.35621  ,  0.53948  , -1.2525
 0.14888  ,  0.17806  ,  0.68418  ,  1.7062
-0.029257 ,  0.19367  ,  0.31023  ,  0.035782
 0.50042  ,  0.44369  , -0.11651  , -0.30781
```

```
-0.40855  ,  0.44103  , -0.47657  , -0.29866
-0.20908  , -0.19897  , -0.30384  ,  0.38789
 0.18508  ,  0.64143  , -0.618    , -0.14658
 0.024929 ,  0.1378   ,  0.041987 , -1.1278
-0.37396  ,  1.0129   ,  0.37305  , -0.48778
-1.1229   ,  0.31191  , -0.68495  , -1.0103
 0.63727  , -0.22982  ,  0.20397  , -1.1838
-0.38747  ,  0.30844  ,  0.85546  ,  1.0522
 0.13913  ,  0.34833  ,  0.19653  ,  0.27499
 0.44288  ,  0.33051  , -0.78574  , -0.76833
```

Figure 13: Context and Target word vector representation

For the new variance presented in the section "Other approaches for the Embeddings Layer" we need to perform by our own the conversion from scalar values to vectors to feed the Neural Network. The word presentation vectors are in a Matrix that will be used to perform the necessary calculation to make the different vector inputs for the different models.

In table 2, we can see the total number of trigrams created from the Penn Treebank dataset to train and test the models in the different structures.

Dataset	%	Trigrams	Total Number of Tokens
Train	70	169,090	240,054
Test	15.1	37,454	
Validation	13.9	33,510	

Table 2 Trigrams create form the Penn Treebank dataset

3.6 Model Structure of the Language models

As part of the augmented models, we plan to incorporate three different variations that fall under the same concept of used the differentiation vector Figure 14. The differentiation is the arithmetic subtraction calculated between two-word vectors resulting in a new vector of the same size dimension. These two-word vectors are taken from the context part of the trigram to predict the target word.

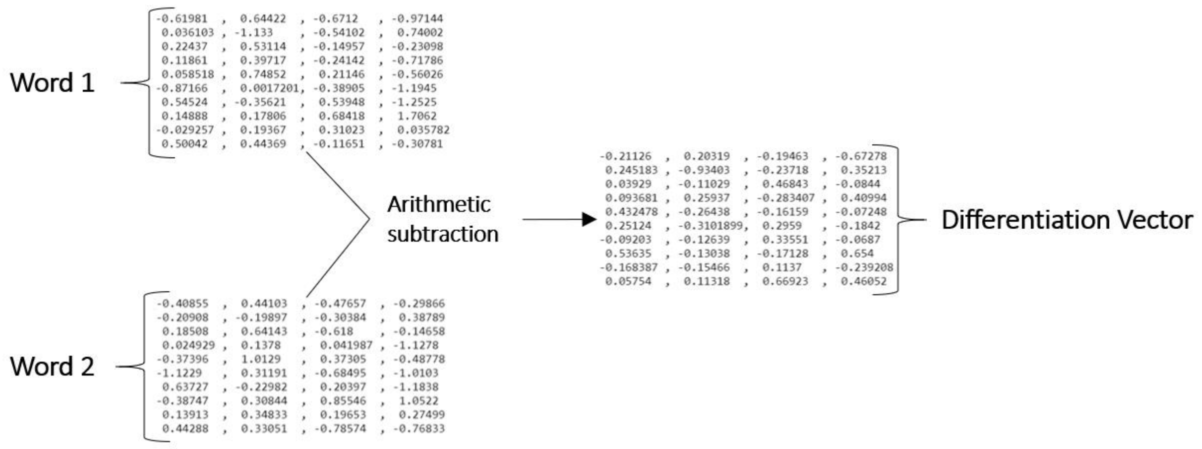


Figure 14: Arithmetic subtraction for the Differentiation vector

3.6.1 Baseline Glove Model

To create the standard structure for the Baseline Glove model, we concatenate the two-words vectors representation of each word in the context by the trigrams to create the input vector to feed the Neural Network figure 15. The pre-processing step creates the input vector into the word embeddings layer.

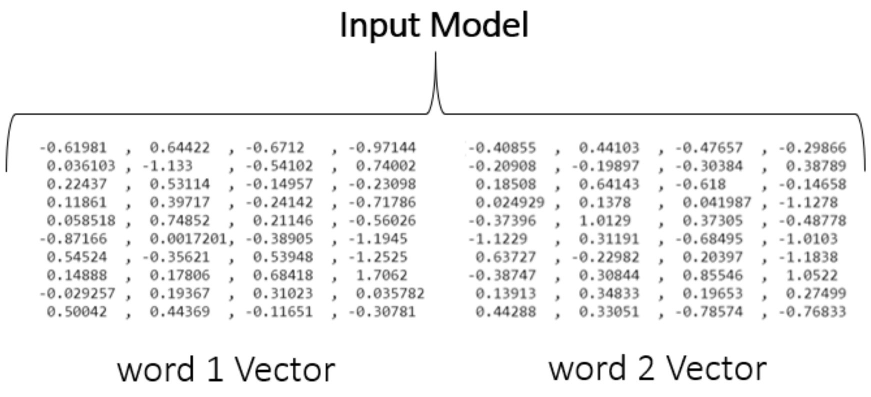


Figure 15: Concatenate input vector the Baseline Glove Model

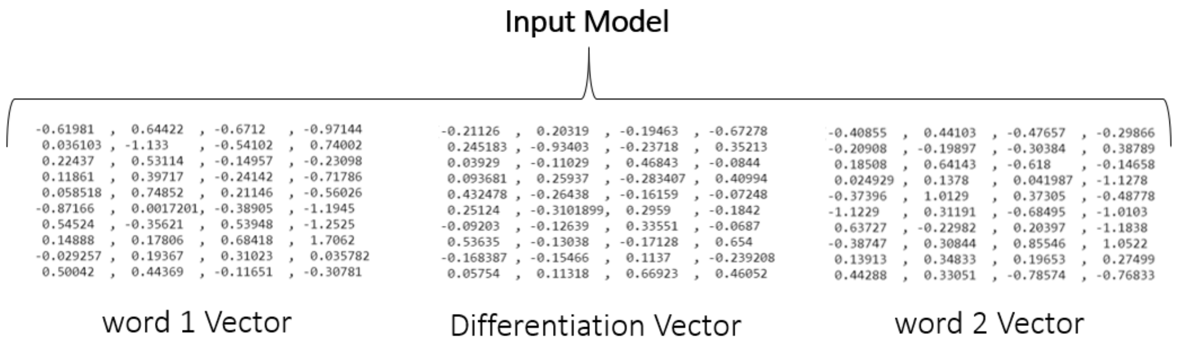
3.6.2 Differentiation Model

The differentiation model boosts the fundamental structure of the Baseline Grove model adding as an addition to the differentiation vector generated by the arithmetic subtraction of the two vectors made the context part of the trigrams.

In this model, we are keeping all the context to predict the next word plus an additional feature. The concatenation of these three vectors is going to generate a new vector with the dimension of the three times the embedding dimension to feed the neural network.

We expect that how the input vector size is bigger than the Baseline Grove model the processing time to train the model could also be an increase.

The vector for this augmented model resulted from the concatenation of the vectors word1 + differentiation + word2; this new vector will be the input of the model. The order of the concatenation does not affect how the information is proceeded internally in the Neural Network, although the Layers and the Hidden Units. It could be to word1 + word2 + differentiation too, figure 16.



```
-0.61981  ,  0.64422  , -0.6712   , -0.97144
 0.036103 , -1.133    , -0.54102  ,  0.74002
 0.22437  ,  0.53114  , -0.14957  , -0.23098
 0.11861  ,  0.39717  , -0.24142  , -0.71786
 0.058518 ,  0.74852  ,  0.21146  , -0.56026
-0.87166  ,  0.0017201, -0.38905  , -1.1945
 0.54524  , -0.35621  ,  0.53948  , -1.2525
 0.14888  ,  0.17806  ,  0.68418  ,  1.7062
-0.029257 ,  0.19367  ,  0.31023  ,  0.035782
 0.50042  ,  0.44369  , -0.11651  , -0.30781
```

```
-0.21126  ,  0.20319  , -0.19463  , -0.67278
 0.245183 , -0.93403  , -0.23718  ,  0.35213
 0.03929  , -0.11029  ,  0.46843  , -0.0844
 0.093681 ,  0.25937  , -0.283407 ,  0.40994
 0.432478 , -0.26438  , -0.16159  , -0.07248
 0.25124  , -0.3101899,  0.2959   , -0.1842
-0.09203  , -0.12639  ,  0.33551  , -0.0687
 0.53635  , -0.13038  , -0.17128  ,  0.654
-0.168387 , -0.15466  ,  0.1137   , -0.239208
 0.05754  ,  0.11318  ,  0.66923  ,  0.46052
```

```
-0.40855  ,  0.44103  , -0.47657  , -0.29866
-0.20908  , -0.19897  , -0.30384  ,  0.38789
 0.18508  ,  0.64143  , -0.618    , -0.14658
 0.024929 ,  0.1378   ,  0.041987 , -1.1278
-0.37396  ,  1.0129   ,  0.37305  , -0.48778
-1.1229   ,  0.31191  , -0.68495  , -1.0103
 0.63727  , -0.22982  ,  0.20397  , -1.1838
-0.38747  ,  0.30844  ,  0.85546  ,  1.0522
 0.13913  ,  0.34833  ,  0.19653  ,  0.27499
 0.44288  ,  0.33051  , -0.78574  , -0.76833
```

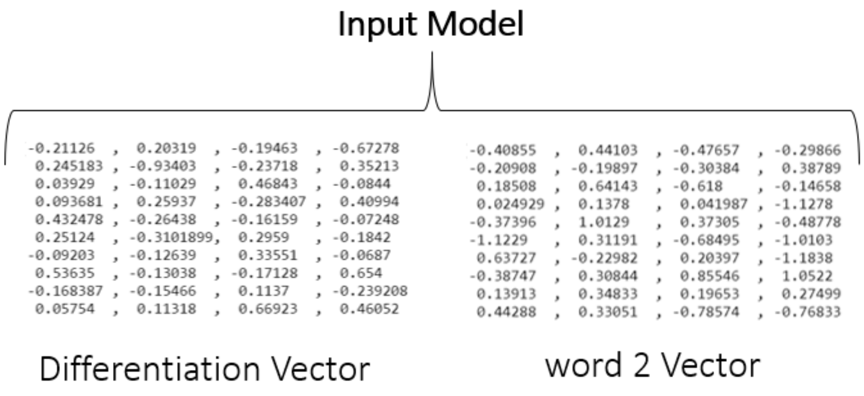
Figure 16: Concatenate input vector the Differentiation model

3.6.3 One-word plus Differentiation Model

In this argument model, we calculate the differentiation vector from the arithmetic subtraction of the two words vector representation from the context. However, this time we only take the resulting differentiation vector and the second-word vector for the context discarding the first word of the context.

Our idea of this implementation is to take advantage of the recurrent feature of the RNN's. This model tries to eliminate redundant information in the input but keep enough context to recognise the words in the vocabulary. The size in the input layer is two times the embeddings dimension.

We expect that the training time is going to be similar to the Baseline Grove model because the input layer of both models has the same size.



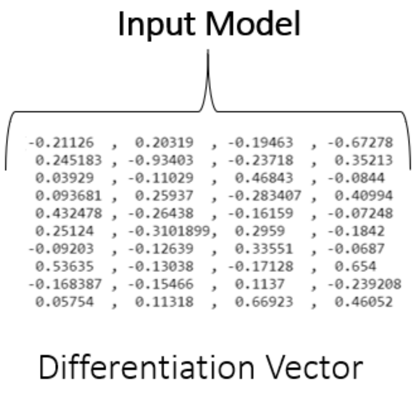
```
-0.21126 ,  0.20319  , -0.19463  , -0.67278      -0.40855 ,  0.44103 , -0.47657  , -0.29866
 0.245183, -0.93403  , -0.23718  ,  0.35213      -0.20908 , -0.19897 , -0.30384  ,  0.38789
 0.03929 , -0.11029  ,  0.46843  , -0.0844        0.18508 ,  0.64143 , -0.618    , -0.14658
 0.093681,  0.25937  , -0.283407 ,  0.40994       0.024929,  0.1378  ,  0.041987 , -1.1278
 0.432478, -0.26438  , -0.16159  , -0.07248      -0.37396 ,  1.0129  ,  0.37305  , -0.48778
 0.25124 , -0.3101899,  0.2959   , -0.1842       -1.1229  ,  0.31191 , -0.68495  , -1.0103
-0.09203 , -0.12639  ,  0.33551  , -0.0687        0.63727 , -0.22982 ,  0.20397  , -1.1838
 0.53635 , -0.13038  , -0.17128  ,  0.654        -0.38747 ,  0.30844 ,  0.85546  ,  1.0522
-0.168387, -0.15466  ,  0.1137   , -0.239208      0.13913 ,  0.34833 ,  0.19653  ,  0.27499
 0.05754 ,  0.11318  ,  0.66923  ,  0.46052       0.44288 ,  0.33051 , -0.78574  , -0.76833
```

Figure 17: *Concatenate input vector for the One word plus differentiation Model*

3.6.4 Differentiation Only Model

In this model, we eliminate the two-word vector representation from the context and just left the word vector differentiation. Reducing, the context given to the network compares to the "One-word plus Differentiation model". This model aims to find if feeding the Neural Network with just the structural information can present a comparable result against the models that have more information in the input.

We expect that the training time will be reduced in comparison to the Baseline Grove model because the vector size only has one time the embedding dimension to feed the Neural Network. It is the vector that contains the arithmetic difference between the two-word vectors of the context. The embedding dimension increase depending on the configuration of the model, this value comes from the vector Size of Glove vector.



```
-0.21126 ,  0.20319  , -0.19463  , -0.67278
 0.245183, -0.93403  , -0.23718  ,  0.35213
 0.03929 , -0.11029  ,  0.46843  , -0.0844
 0.093681,  0.25937  , -0.283407 ,  0.40994
 0.432478, -0.26438  , -0.16159  , -0.07248
 0.25124 , -0.3101899,  0.2959   , -0.1842
-0.09203 , -0.12639  ,  0.33551  , -0.0687
 0.53635 , -0.13038  , -0.17128  ,  0.654
-0.168387, -0.15466  ,  0.1137   , -0.239208
 0.05754 ,  0.11318  ,  0.66923  ,  0.46052
```

Figure 18: *Concatenate input vector for the Differentiation Only model*

3.7 Apply Grid Search to find best Parameters

The purpose of performing the Grid search if to find the hyperparameters that allow us to create the optimal model and give us a better understanding of how significant is adding additional structure information as an input to feed the models. These findings are possible by performing an exhausting search in all the possible combinations of the hyperparameters.

The hyperparameters that make up the RNN models have splinted in 2 sets the Fixed parameters and the Grid Search paraments. For the Grid Parameters the range of values whee selected taking as a reference the results found in the literature review. It is of our primary interest find parameters that allow creating a robust model in which we can apply more exhaustive testing to measure the impact of our augmented models.

3.7.1 Fixed Parameters

- Number of Epochs: [15]
  An epoch is when the complete dataset is passed forward and backwards through the neural network one time. In each forward pass the error is computed, in the backward pass, the model updates the weights of the hidden neurons.

- Mini-batch size: [16]
  The mini batch gradient descent is commonly using in Deep Leaning (DL) to calculate the error model and update the coefficients (Brownlee, 2017). It splits the training data into batches of normalising sizes of (8, 16, 32, 64) depending on the Neural Network needs. In this implementation a sum of the gradient is calculated over the Mini-batch to get an average to update the coefficients, it helps to generalise them and reduce the variance.

- Number of Layers: [2]
  One of the main characteristics of the Artificial Neural Networks (ANN) structures is the possibility of go depth, adding more layer to the model. A number of layers higher than two can significantly increase the computation cost to process the vector across all the layer and contribute to the problem of vanishing gradient. Some research has provided evidence that the number of layers has more influence on the performance of the network than the number of hidden units in each layer (Graves et al., 2013)

- Output Layer: [1000]
  The size of the vocabulary size of the training set that contains 1000 different words is used as the last layer of our models. Each model is going to share this characteristic to calculate the perplexity for each possible word in the Vocabulary.

3.7.2 Grid Search Parameters

- Hidden Layer Size: [200, 400]
  The prmary parameter to test; hidden layer size defines the number of hidden units per layer in each cell type (RNN, LSTM, and GRU). A big number can result in over-fitting the training set, and a lower number can predict the outcome on under-fitting (Panchal et al., 2011). Over-fitting appears when the Network has more neurons than necessary, Under-fitting appears when the Network has few hidden units in comparison with the complexity of the network, it can be interpreted that the network does not understand the problem (Panchal et al., 2014). To avoid these two scenarios, we want to keep this parameter in the rage between 200 to 400; a big number can increase the computational cost in the training phase too.

- Embeddings Dimension [50, 100, 200, 300]
  The Vector sizes contained in the In Glove: Global Vectors for Word Representation are in the dimension of 50, 100, 200 and 300. These paraments will take as word embeddings dimensions. It is a primary interest find out how is the impact of this vector size in the model results.

- Dropout [0, 20]
  As a regularisation technique, we applied Dropout; it was introduced by (Srivasta et al. 2014) The idea is to select a percentage of neurons into the layers randomly; these neurons are going to have ignored them during the training phase, these neurons are "dropped out". It means that the contribution of this neurons during one forward-pass and backwards steps to calculate the error in the neural network used to update the internal weights is ignored. (Zaremba et al. 2014) Apply dropout usefully on LSTM. The Dropout value represents the probability that a neuron is not dropped. Its implementation helps the model to generalise better and avoid overfitting the train set.

3.8 Performance Measure

The main guideline is to find the model with lower perplexity. We want to contrast the base model against the other augment models. We train the model with the training set over 15 epochs, an epoch is when the complete train set pass forward and backwards through the neural network one time, during each epoch the model generate two perplexity values one for the train set and other for the test set, we document these results to record the model performance in the training phase.

Another critical factor is the execution time to evaluate the computational cost of each model; the measure was made in seconds, taking the processor time from the model start to be trained until the model complete the total number of epochs, for an understanding purpose, the tables display this result in minutes. The goal of this measure is to give us a clear idea on how more complex models affect the training face and how is the performance in the results. We can determinate a balanced model that can show good results in a reasonable time in the training phase.

During the training phase, our models we realised that after an N number of epochs the model tends to overfit the training set, Another metric is to find in which epoch the model performs well in both sets before it starts to overfit.

3.9 Testing the significance of the Selected Models

After performing the grid search, we found the parameters that create the model that show a significant change in the perplexity results. The Peen Treebank dataset contains a validation set that we can use to measure our performance with the literate. However, for this project, we want to ensure that the result that we have are significant, and we want to create more metrics to compare the performance of the base model against the models that are implementing the differentiation as additional feature information.

3.10 Significance Test: Wilcoxon Signed Rank Test

To check the significance of our results against the base model, we did not assume of normality on the results. We did not apply the commonly used Pair T-Test. Instead, we opted for the non-parametric Wilcoxon Signed Rank Test. This test is typically used to check the significance of the difference in "before" and "after" conditions.

The before condition is the perplexity measure generated per word by the base model, the after the condition is interpreted as the perplexity generated per word for the different augmented models.

Wilcoxon Signed Rank test analyses the matched-pair data (Woolson, 2008). To implement this test, there are a few assumptions that have to follow; these assumptions are discrete below. Otherwise, we need to adopt another test.

- The dependent variable (x) must be continued and measure distinguished with values of th decimals.
- The paired values are coming from a continues is random population.
- The pared values are coming from the same population.

The absolute values are calculated and ranked from the paired values. The statistics generated by summing the negative or positive paired ranked values.

We need two hypotheses to start the test:

Null hypothesis $h(0)$: The difference generated from the pair values has an approximation around zero.

Alternative hypothesis $h(1)$: The difference generated from the pair values does not follow an approximation around zero.

We define W value of 0.05 for the two-sided p-value that the Wilcoxon Signed Rank Test return. If the p-value generates by the test is less than this threshold, we can reject the null hypothesis.

4 Primary Grid Search Results

This section contains the result of the experiments applied to the grid search. The experiments are performed using a word embedding dimensions of 50, 100, 200 and 300. The complete tables generated by the experiments can be looked up in the appendix B. As a result of the Grid search, we found that the embedded dimension of 300 helps to create the best experiments of each model, this dimension size of 300 and the dropout value setup to 0 were ones of the critical parameters to make up the next experiments on the significance test, the configuration showed the significant chance in the perplexity generate by all the experiments.

4.1 Results of Internal against external Embeddings layer

To start the experiments, we made the comparison between how the standard model implement the embeddings in the Natural Language models (Word Embeddings PyTorch, 2018) we explain this structure in deep in the section in the section 3.5.1. By nature, Recurrent Neural Network's training phase requires so many resources to train the model. Our idea is to optimise the model much as we can, moving out the layer that performs the embeddings function from inside the model to an additional step in the pre-processing phase using a custom function.

We ran all the RNN types (RNN, LSTM and GRU) in all the model types (Glove, Differentiation, One Word Plus Differentiation, Differentiation Only), we processing time, and Perplexity generated by the models running in 15 epochs. For this experiment, the aim is to measure how much the Training time chances between both architectures, table 3.

For this experiment, the hyperparameters are set up as **Epochs**: 15, **Embeddings dimension**: 50, **Hidden Units**: 200 and **Dropout**: 0

Type	Model	Internal Layer	Pre-processing	%
RNN	Glove	40.9	28.56	30.17
LSTM	Glove	53.03	38.55	27.31
GRU	Glove	39.19	37.88	3.34
RNN	Differentiation	45.84	31.18	31.98
LSTM	Differentiation	56.84	39.49	30.52
GRU	Differentiation	50.84	37.88	25.49
RNN	One Word + Diff	45.64	28.53	37.49
LSTM	One Word + Diff	55.83	39.55	29.16
GRU	One Word + Diff	51.55	37.45	27.35
RNN	Differentiation Only	45.83	29.44	35.76
LSTM	Differentiation Only	47.46	38.86	18.12
GRU	Differentiation Only	54.85	37.76	31.16
			Total Average	**27.32**

Table 3: Results for internal against prepossessing embeddings layer.

As a result, it is gratifying to see that that modification that we propose show significant results in comparison of the standard implementation. In average we reduce the processing time by 27.32 % in the 12 different models. Showing the highest improvement in the model RNN & One Word Plus Differentiation with 37.49% and the lowest difference in the model GRU & Baseline Glove with just 3.34%, in the rest of the models, the reduction was over 25% in most of the cases.

The results showed that this architecture of prepossessing the embeddings outside the model turned out to be an outstanding performance. This architecture is going to be implemented in the rest of other experiments to perform the Grid Search and find the best parameters. We plan to run 192 different experiment with all the Grid search model variation this implement is an architecture that helps us to save in average over 27% in the training time will be very beneficial to complete our project on time.

4.2 RNN Grid Search Results

We start discussing the results for the Baseline Glove model in table 4. We did the experiments using a simple RNN cell. We used the same fixed parameters, the Learning rate setup to 0.1. We expect that the performance of this model was weak. Because the RNN structure is not as well other types of cell to deal with Long sequences. We covered this on section 2.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	85	300	400	0	53.96	6.806	15	53.396	3
2nd	37	300	200	0	43.23	12.007	15	54.13	3
3rd	13	100	200	0	42.48	13.519	15	55.176	4
4th	169	200	400	0.2	81.15	13.248	15	55.503	3
5th	25	200	200	0	46.25	13.39	15	55.637	4
16th	97	50	200	0.2	35.92	24.823	15	61.887	6

Table 4: RNN – Baseline Glove Model outstanding experiments

It is clear that with a higher embeddings dimension and a bigger number of hidden units, the model produces lower perplexity result. The best model for the Cell RNN baseline Glove is the model-85 with the lower perplexity values of 53.39 in the epoch numb 3.

As we expect, the RNN cell overfits the Train set to quick, showing the best perplexity result for the Test set between the epochs three and five in most of the different configurations. All the models are running until 15 epochs it means that the RNN cell did not manage long dependencies properly into the training set, as the literature says (Anish Singh, 2017), one of the reasons could that the gradients exploded exponentially a typical experience in the RNN architectures the long dependency problem.

The dropout values appear to not contribute to the creation of a better model; the best three different configuration has the dropout values set up to 0. A model with a higher dropout value model-169 produces a higher perplexity that a model with a higher embedding size model-85, also we can see how the droop value increases the training time like in the model-169 against model-73 with the same configuration except for the dropout value.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	88	300	400	0	79	7.726	14	54.642	3
2nd	40	300	200	0	29.48	13.178	14	55.775	4
3rd	76	200	400	0	53.74	7.622	15	55.817	3
4th	64	100	400	0	80.52	8.225	15	56.43	4
5th	16	100	200	0	43.31	14.268	15	56.521	3
16th	148	50	400	0.2	79.87	15.896	15	61.955	2

Table 5: RNN – Differentiation Model outstanding experiments

Table 5 exhibits the results of the Differentiation model implemented in RNN structures. These are the results for our first augmented model; we found the best model for this Grid search with a perplexity of 54.642 model-88, it does not overcome the best model of baseline Glove model-85.

The model with larger sizes embeddings and hidden units shows a better perplexity result, in the other hand we can see how this also contribute to increasing the training time dramatically from 29.48 min in the model-40 to 79 mins in the model-64 even when the results in the perplexity are very close to each other 55.775 and 56.43 respectively

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	43	300	200	0	28.56	13.5689	13	54.7393	3
2nd	175	200	400	0.2	171.56	13.6184	15	55.4471	3
3rd	91	300	400	0	53.96	7.5035	15	55.7516	2
4th	31	200	200	0	46.22	14.0701	14	56.4347	3
5th	67	100	400	0	78.03	8.2963	15	56.4388	4
16th	103	50	200	0.2	35.96	25.2319	15	60.6197	3

Table 6: RNN – One word plus differentiation Model outstanding experiments

The One word plus differentiation model results continue to exhibit in the inconsistency of RNNs (table 6). For example, the best model for this configuration model-43 outputs displays a higher perplexity that the baseline Glove equivalent. For the first time, we found a model with a Dropout value different setup to zero into the first three places for these experiments model-175.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	94	300	400	0	51.46	7.259	15	57.345	3
2nd	46	300	200	0	33.54	13.156	15	58.114	4
3rd	34	200	200	0	45.17	14.684	15	58.414	3
4th	190	300	400	0.2	77.74	13.454	15	58.613	2
5th	82	200	400	0	78.62	8.183	15	58.645	3
16th	106	50	200	0.2	33.94	27.332	14	66.927	5

Table 7: RNN – Differentiation Only Model outstanding experiments

The last augmented Differentiation Only model results are exhibited in table 7. We got the best model with a higher embeddings dimension and hidden units, model- 94. At this point, it is evident the inability of the RNN architectures to handle long sequences, this problem of the RNN only allow us the find the model that generates the lower perplexity in early epochs between 2 and 4.

In figure 19 exhibits the models with the best performance. We can see the problem of the long sequence is too evident, the model in the training set cannot generate a model with lower perplexity beyond early epochs between 2 and 4. Resulting in exponential values in the perplexity generate.

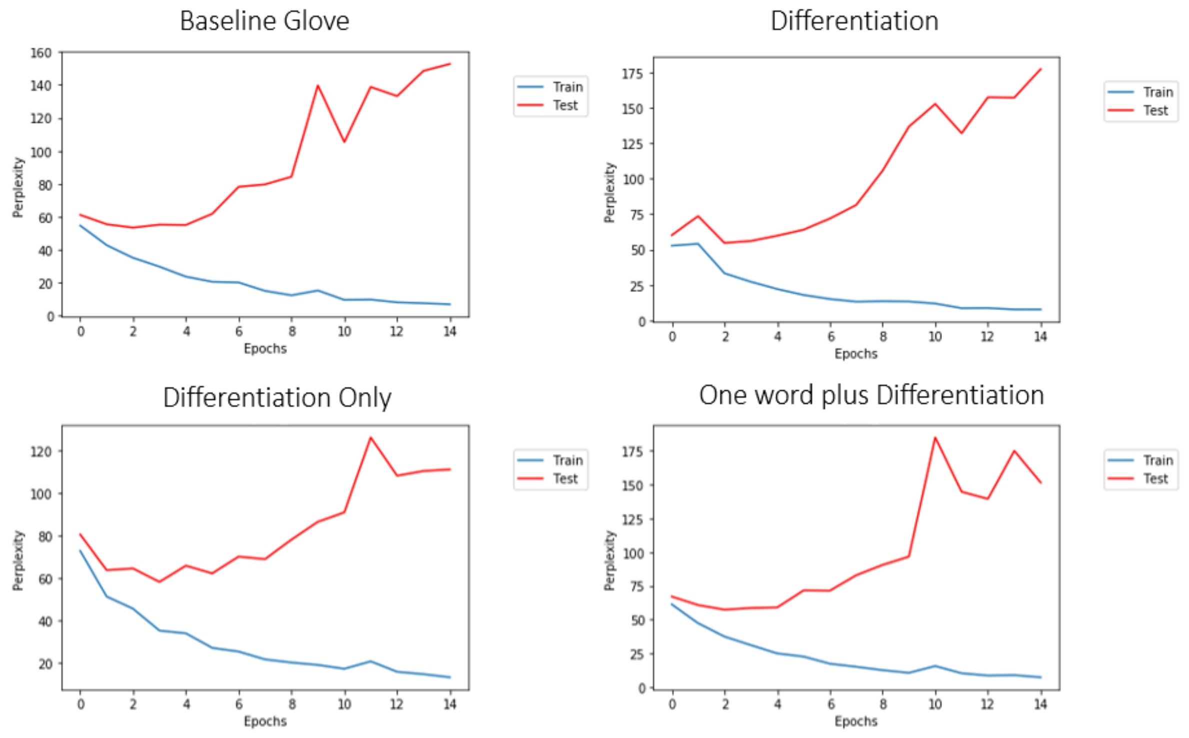


Figure 19: RNN best models' performance

4.3 GRU Grid Search Results

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	39	300	200	0	35.04	12.671	15	52.254	7
2nd	27	200	200	0	57.7	15.198	15	52.515	7
3rd	135	300	200	0.2	55.67	20.565	15	53.435	7
4th	123	200	200	0.2	56.25	22.103	15	53.611	9
5th	15	100	200	0	50.4	17.413	15	53.788	7
16th	147	50	400	0.2	141	17.233	15	57.874	7

Table 8: GRU – Baseline Glove Model outstanding experiments

The GRU structure in the baseline Glove model (table 8) predict perplexities that are lower than the RNN baseline Glove model. A higher embeddings dimension and hidden units look to be a constant in the contribution of creating models with low perplexity.

The main difference with the RNN structures is that GRU manages better the long sequences in a better way, we can see how the best epochs for the training set are between the 5 and 9.

Finding the best model variation with a perplexity of 52.254 for the model-39, and this time showing the best execution time in the training phase with 35.04 mins.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	42	300	200	0	34.36	11.595	15	52.043	4
2nd	186	300	400	0.2	198.9	11.056	15	53.361	6
3rd	30	200	200	0	57.9	13.848	15	53.445	6
4th	138	300	200	0.2	57.54	19.184	15	53.531	6
5th	78	200	400	0	91.35	6.811	15	53.73	5
16th	150	50	400	0.2	140.9	15.45	15	57.204	6

Table 9: GRU – Differentiation Model outstanding experiments

The effect of adding additional context with the model GRU -Differentiation show better results than the Baseline Glove model for the first time (table 9) in the model-42 with a perplexity of 52.043. The rest of the other variation does not approximate too much to this value the closest one is the model-186 with a Perplexity of 53.361.

We can deduce that one of the reasons can be that the best epoch for the training is between four and six in comparison the baseline with the epoch between five and nine.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	45	300	200	0	32.9	12.196	15	52.195	6
2nd	33	200	200	0	56.17	13.781	15	52.332	5
3rd	141	300	200	0.2	111.6	19.707	15	52.603	7
4th	93	300	400	0	91.51	6.75	15	52.654	4
5th	129	200	200	0.2	45.68	21.484	15	52.901	7
16th	57	50	400	0	129	9.817	15	57.175	7

Table 10: GRU – One word plus Differentiation Model outstanding experiments

The general effect of adding an extra feature in the One word plus differentiation model appears to have a positive influence in reducing the perplexity (table 10). Following that pattern that we saw more consistent a higher embeddings dimension with high number of Hidden unites size plus a dropout value equal to zero, the best model for this Grid search is the model-45 with a better perplexity that the best model of the Baseline with 52.195 founded in the epoch number 6 in the testing set.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	96	300	400	0	88.88	7.273	15	55.9	5
2nd	48	300	200	0	32.56	13.789	15	56.052	6
3rd	36	200	200	0	56.4	15.126	15	56.309	6
14th	12	50	200	0	50.35	19.943	15	61.529	6
15th	108	50	200	0.2	32.62	27.866	15	63.404	8
16th	156	50	400	0.2	129.5	15.779	15	64.444	5

Table 11: GRU – Differentiation Only Model outstanding experiments

The augmented model Differentiation Only is created with the differentiation vector generated from the arithmetic subtraction of the two-word vector representation does not show any improvement against the Baseline Glove models with a perplexity of 55.9 in the best model-96 of this Grid search, table 11.

These Gird search results showed the larger perplexity values generate for this model implementation starting with 55.9 by the model-96 and going up 64.444 in the model-156. We can begin deducting that this model implement does not improve to get a good prediction for the words in the vocabulary.

Figure 20 shows the results for the best models in GRU structures. This time GRU manages in a better way the long sequences in the Training set before the perplexity generates by the model start increasing in epochs between 5 and 9.

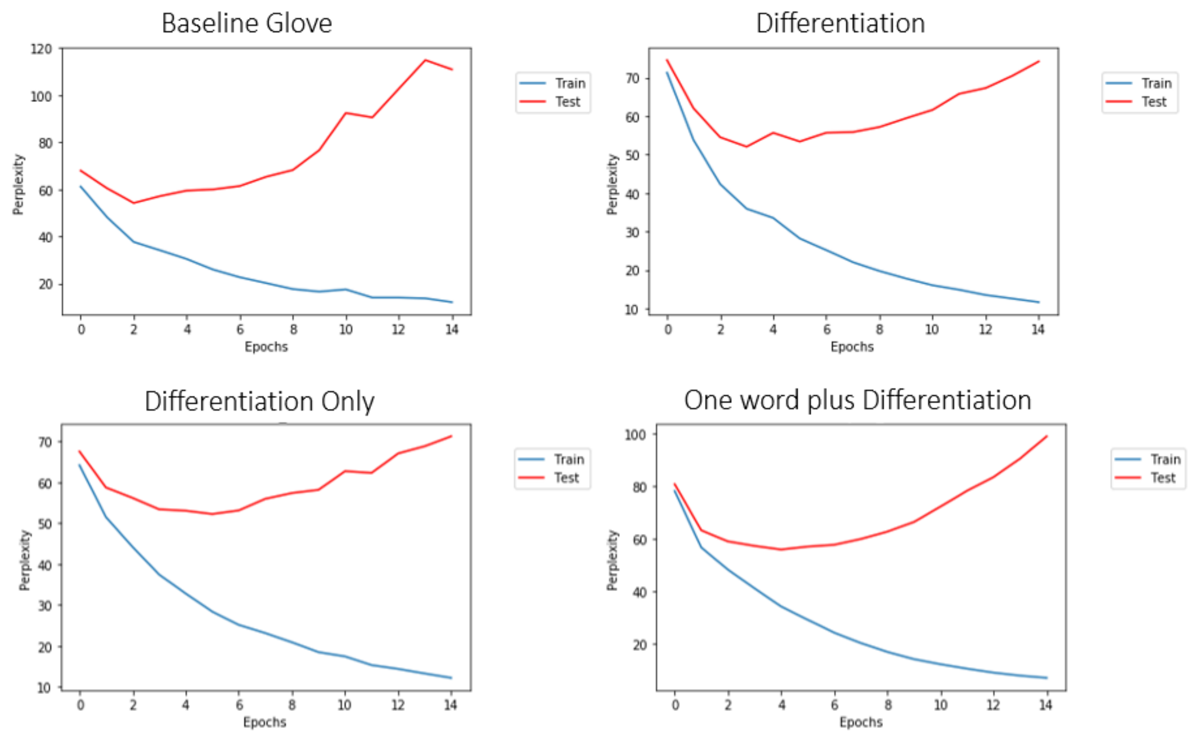


Figure 20: GRU best models' performance

4.4 LSTM Grid Search Results

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	38	300	200	0	65.04	19.686	15	51.724	8
2nd	86	300	400	0	97.09	14.249	15	51.847	8
3rd	182	300	400	0.2	180.9	20.528	15	52.153	9
4th	134	300	200	0.2	58.82	27.24	15	52.277	11
5th	122	200	200	0.2	57.12	29.365	15	52.577	11
16th	146	50	400	0.2	139.3	29.472	15	56.308	12

Table 12: LSTM – Baseline Glove Model outstanding experiments

As expected, the LSTM base model Glove produces the lower perplexity values (table 12). However, the merging of difference is surprising due to GRU's potentially produce comparable results as the literature mentions (Chung et al., 2014). The results are consistent for all the embedding dimension sizes. However, tendencies show that the Dropout value of zero produces the two best models that for the first time in the experiments the perplexity falls below 52, the model-38 with 51.724 and model 86 with and 51.846.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	185	300	400	0.2	237.1	16.822	15	52.087	7
2nd	89	300	400	0	101.7	10.578	15	52.154	7
3rd	77	200	400	0	101.4	13.276	15	52.318	8
4th	29	200	200	0	61.16	19.737	15	52.667	7
5th	41	300	200	0	36.03	16.858	15	52.694	8
16th	149	50	400	0.2	136.4	26.474	15	56.248	12

Table 13: LSTM – Differentiation Model outstanding experiments

The Differentiation model was not able to generate results that beat the results of the experiment of the Baseline Glove model (table 13). Something to notice is that the dropout value contributed to produce the best model for this Grid Search in the model-185 with 52.087, but the processing time needed to train the model against the rest of model is considerable with 237.1 mins, the 2nd model outperformed the other models with varying values.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	44	300	200	0	34.25	17.805	15	51.347	6
2nd	92	300	400	0	97.42	11.707	15	51.43	8
3rd	32	200	200	0	58.86	20.215	15	51.926	10
4th	140	300	200	0.2	127	25.145	15	51.956	10
5th	188	300	400	0.2	236.5	17.986	15	51.992	7
16th	152	50	400	0.2	136.2	25.772	15	55.618	12

Table 14: LSTM - One word plus Differentiation Model outstanding experiments

The One word plus differentiation model (table 14) produce better perplexities that their baseline Glove equivalent, where this is showed in the results. A dropout value of zero processes low test perplexity in the best model-44 with 51.347. This model shows outstanding results on the five first experiments perform below 52 in perplexity.

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	47	300	200	0	32.99	21.801	15	54.489	9
2nd	143	300	200	0.2	32.81	29.656	15	55.126	11
3rd	95	300	400	0	93.46	16.472	15	55.299	8
4th	191	300	400	0.2	137	21.659	15	55.337	11
5th	83	200	400	0	136.6	17.18	15	55.47	8
16th	107	50	200	0.2	34.06	39.154	15	61.79	12

Table 15: LSTM – Differentiation Only Model outstanding experiments

The Differentiation Only model also shows a lower performance in the Perplexity generation in LSTM structures implementation. The best model-47 with a perplexity of 54.489 is not close its baseline Glove equivalent. The size of embeddings and Hidden Units show to be a constant in the contribution of creating a better model. However, for this run, a smaller hidden unit's size shows better results that use a higher size as the model-47 and model-143 display in the (table 15).

It is evident how the LSTM structure deals better with the long sequences allowing to run the model until 8 to 12 epochs before to generate overfitting on the Training set. The results for the best model are exhibited in figure 21.

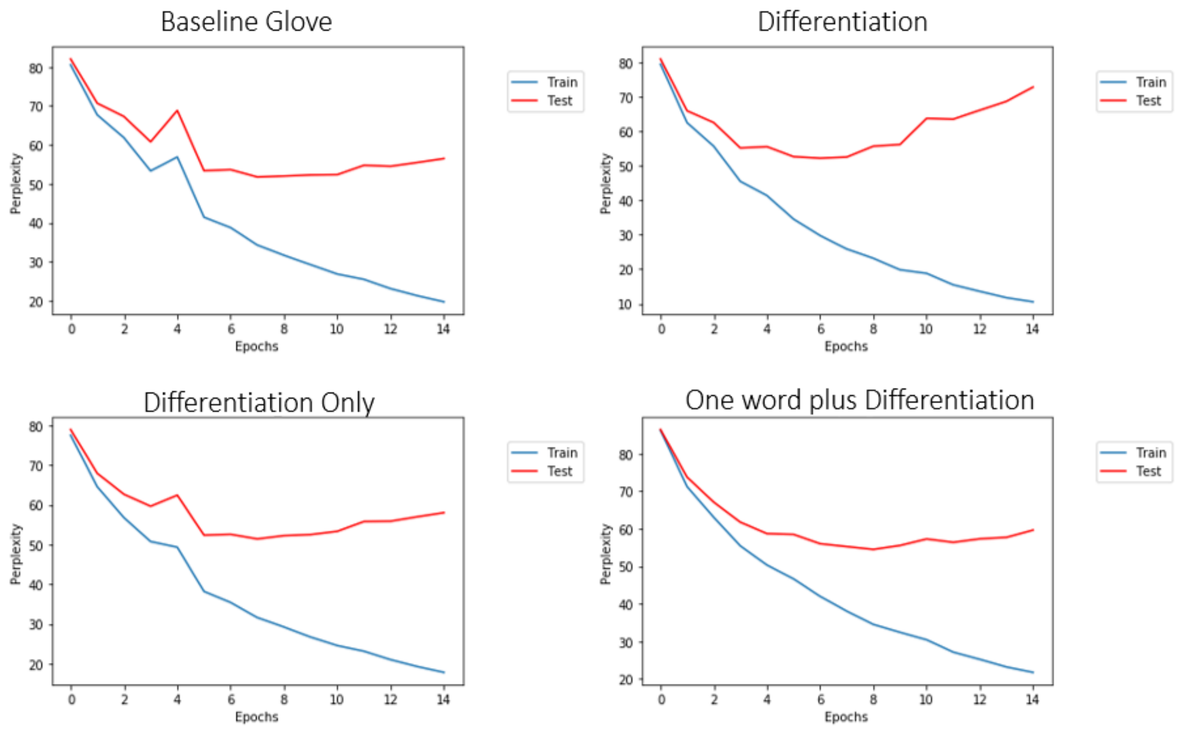


Figure 21: LSTM best models' performance

4.5 Conclusion form the Preliminary Grid Searches

As we expected the RNN grid search show a poor perplexity value in comparison to GRU and LSTM cells. The volatility showed by RNN to overfit the training set in early epochs does not allow to create a model that competes with its contra its equivalents.

As a result, from our given parameters and results provided by the grid search, we can conclude that adding structural information to a simple RNN, the problem of dealing with long sequences is most evident. These results help us confirm the literate review that mentions the behaviour of the RNN when dealing whit this kind of long sequence problems.

Our objective is to focus more on the comparison between the GRU and LSTM architectures, which provide better perplexity results and more consistency values generated.

Our primary goal is to investigate whether adding addition structural information can improve the baseline Glove model. For some hyperparameter, both cells showed improvements in using the advantage of the additional differentiation vector. LSTM have shown consistent results with a range of a high number of hidden units; the dropout values set up to zero tend to have a more significant effect in generating a model with lower perplexity. GRU appears to take advantage of a hidden unit’s size smaller than the used in LSTM to create the best model, GRU also improves the model with the use of the additional feature.

The length of training time between GRU and LSTM show to be very similar. The GRU has less complexity in its internal structure; we have the intuition that it will contribute to lower training time. It is reflected in some grid results were the LSTM train time has smaller for some experiments.

It is mostly that evidence that the Experiments with a small dimension size of 50, 100 perform poorly in a comparison of the model that implements a higher value. Showing that this one the main hyperparameter to take care to create a good model, this behaviour was the same in the results for the three different cells.

The results give us a positive intuition of the use of additional structure information contribute to the creation of models with lower perplexity. It is denoting that comparison between the different augmented models against each model type cannot be done, the primary purpose is to take the best of this model and check if they can beat the baseline Glove model.

The next step is to select the best model that perform better in each Cell type, play a significance test to validate how much are we improving with the relation of the Baseline Grove model. Table 16 contains the selected models.

Cell	Model	Embeddings	Hidden Units	Dropout	Numb
RNN	Glove	300	400	0	85
	Differentiation	300	400	0	88
	One Word plus Differentiation	300	200	0	43
	Differentiation Only	300	400	0	94
LSTM	Glove	300	200	0	38
	Differentiation	300	400	0.2	185
	One Word plus Differentiation	300	200	0	44
	Differentiation Only	300	200	0	47
GRU	Glove	300	200	0	39
	Differentiation	300	200	0	4
	One Word plus Differentiation	300	200	0	45
	Differentiation Only	300	400	0	96

Table 16: Candidate Experiments to perform the significance test.

5. Results for Selected Models

For this section, we apply a significance test measure on the selected model form the table 16. The Wilcoxon is going use the best of the augmented models that show a lower perplexity compared to the best architecture of the baseline models.

With the usage of Wilcoxon Signed Rank Test, we want to determinate how significant are the perplexity results showed by the augmented model or if they are not significant by any chance. We create a Null hypothesis to determinate the guideline of the Wilcoxon Test. We accept the Null hypothesis if the p-value >= W value and reject the Null hypothesis if the p-value < W value.

As a result, rejecting the Null hypothesis means that the augmented model has a significant effect on the perplexity in otherwise the augmented model does not improve enough the achievement by the Baseline Grove model.

5.1 Applying significance Test to the Augmented models

We are going to perform the significance test just on the models that overcome in perplexity results of the Baseline Glove model. The comparison is going to be done in the models using the same Type of Cell (RNN, GRU, LSTM). The significance test is done in two model to get the p-value, we use the baseline model and the augmented model. We ran each model in the testing set and the validation set; we got the perplexity per each word in the sets.

5.1.1 RNN Cell Type

The results for the best models are exhibited in table 17, Starting with the results for the RNN Cell type, we can see that any of the Augment models can beat the perplexity generate by the baseline Model with the closest one generates by the model Difference Augment model. Follow our scoop we do not perform a significance test on this Cell Type.

Cell	Model	Embed-dings	Hidden Units	Dropout	Numb	Time	PLL Train	Best epoch	PPL Test	Best epoch
RNN	Glove	300	400	0	85	53.96	6.806	14	53.396	2
	Differentiation	300	400	0	88	79	7.726	13	54.642	2
	One Word + Diff	300	200	0	43	28.56	13.569	12	54.739	2
	Differentiation Only	300	400	0	94	51.46	7.259	14	57.345	2

Table 17: RNN Grid search best Model

5.1.2 GRU Cell Type

Continuing with the Result on the GRU cell, we have that two model that beat the perplexity of the baseline model, the One word plus Differentiation and the Differentiation model with values of 52.195 and 52.043 respectively.

We are going to take these models model-45 and model-42 to perform a significance test against the baseline Glove model model-39 and discard the rest of the other model because they do not outperform the aim to beat the baseline model perplexity.

We took the Baseline, and the two models the Differentiation and the One word plus differentiation model, we use the Test and Validation sets to confirm that the result generate consistent in different datasets. With this result, we can see that the Perplexity generates by the models a little higher on the Validation set that the Test set. However, the results of the Augment models are performing better than the base model; this is the consistency with what we expect to see, the augment models are overcoming the baseline models in unseen data again, table 18.

Cell	Experiment	Embed-dings	Hidden Units	Dropout	Numb	Time	PLL Train	Best epoch	PPL Test	Best epoch
GRU	Glove	300	200	0	39	35.04	12.671	14	52.254	6
	Differentiation	300	200	0	42	34.36	11.595	15	52.043	4
	One Word + Diff	300	200	0	45	32.9	12.196	14	52.195	5
	Differentiation Only	300	400	0	96	88.88	7.273	14	55.9	4

Table 18: GRU Grid search best Model

We perform two significance tests, one of the Differentiation models against the Baseline model and other of One word plus differentiation model against the Baseline model. We ran each model in the test and validation set to calculate the perplexity value per each word in both sets.

The next step was to compare these results to get how significant are the difference (table 19). As a result, the p-values on the Test set was 0.0627, and for the Validation, the set was 0.0766, for the test done between Baseline vs One Word Plus Differentiation. Moreover, the p-values on the test set were 0.1739 and, in the Validation, set was 0.0701 between Baseline vs Differentiation. Following out the Null hypothesis we are getting that these results are now below the 0.05 and therefore they are NOT significant, a result we cannot reject the Null hypothesis.

Cell	Model	Embed-dings	Hidden Units	Dropout	Numb	PPL Test	PPL Valid	Test	Validation
GRU	Glove	300	200	0	39	52.254	52.344		
	Differentiation	300	200	0	42	52.043	52.157	0.173900	0.070174
	Differentiation Only	300	200	0	45	52.195	52.285	0.062782	0.076690

Table 19: GRU Significance Test on augmenting models that outperform baseline model

5.1.3 LSTM Cell Type

Analysing the results on the LSTM Cell, we found that as the results in the GRU cell the model One word plus differentiation was the only augment model that show bellow perplexity that its equivalent Baseline model. It is important to mention how the Augment model shows better results early in the epoch number 6 compeer to the best result in the baseline model epoch number 7. Also, the training time for our augment model was considerably below with 34.25 min compared to 65.04 of the baselines.

The performance shown by this experiment model-44 is impressive in comparison to the rest of the experiments, the perplexity value generated by this model is the lowest in all the Grid search across the different Cell type too, table 20.

Cell	Experiment	Embed-dings	Hidden Units	Dropout	Numb	Time	PLL Train	Best epoch	PPL Test	Best epoch
LSTM	Glove	300	200	0	38	65.04	19.686	14	51.724	7
	Differentiation	300	400	0.2	185	237.1	16.822	14	52.087	6
	One Word + Diff	300	200	0	44	34.25	17.805	15	51.347	6
	Differentiation Only	300	200	0	47	32.99	21.801	14	54.489	8

Table 20: LSTM Grid search best Models

We took the perplexity values generated by this augmented model per each word in both the test and validation sets and compared with its equivalent baseline model doing the same process. As a result, we are getting p-values for a test set of 0.610 and the validation set of 0.519. Following our premise that the p-value has to be below 0.05 to be considered significant. We can conclude that this model does not archive this aim and therefore we can reject the Null hypothesis, table 21.

Cell	Model	Embed-dings	Hidden Units	Dropout	Numb	PPL Test	PPL Valid	Test	Validation
LSTM	Glove	300	200	0	38	51.724	51.814		
	Differentiation Only	300	200	0	44	51.347	51.437	0.61047	0.51930

Table 21: LSTM Significance Test on augmenting models that overperform baseline model

6. Discussion: Answering the Research question and Results analysis

In this project, we aimed to investigate the following question: *Can the difference calculated between words from a pre-trained word vector representation "Glove" improve word prediction of Language models with Recurrent Neural Network structures?*

Regarding our results, the answer is no, even though we can reduce the perplexity of the augmented models on the LSTM and GRU structures, these results showed not be significant against the perplexity generated by the Baseline model according to the Significance Test that we perform on the selected models.

The cornerstone of our project was the word vector representation "Glove", the vectors obtained from this unsupervised learning algorithm are based on word cooccurrences that can be transposed to a subtraction of the word vector space and give us a semantic distance between these words.

We wanted to find whether a Difference vector calculated form two-word vectors representation can be used as an alternative to the Baseline Glove model. We implement this new vector on three different models, the model Differentiation, the model One word plus differentiation and the model Differentiation Only; we explain in depth the internal structure of each model and how these models manage the creation of the input vector to feed the Neural Networks in section 3.

After running all the experiments, we found that a larger embeddings dimension size contributes to creating a model with lower perplexity. All the best model uses the embedding size of 300 in the three different cell types (RNN, LSTM, GRU).

The models implemented in RNN structures show how the Long Dependency problem affect the work prediction of the model's overfitting the training set in early epochs during the training phase. Any of our augmented models can beat the baseline model in the perplexity results. The implementation of the model in this structure helps us to corroborate how is the behaviour of the RNN structure in the sequence of data. Also, we found that the N-Gram models that are represented by the base model are very efficient by itself contributing to many of the state-of-the-art results in this field.

We got that two of the three models implemented using RNN provides close results compared to the baseline model generating perplexity below 55, not so far from the 53.396 by the best experiment of the baseline model. On the other hand, the model Differentiation Only, a model that only feed the network with the differentiation vector, shows perplexity results above 58 in most of the experiments with the best result of 57.345, we can see the first signs that the information provided to the network for this model is not enough to compete with the baseline models.

Continuing with the models implemented in GRU structures, the best models in the grid search took more advantage of a small hidden units' size, with three of the models using 200 sizes and just one the Differentiation Only model using a size of 400. Two of our augment models can beat the perplexity generate by the baseline model, the Differentiation model and the One word plus differentiation model, with a similar configuration in the embeddings dimension, size of the hidden units and dropout value.

Also, these two models found the best perplexity value for the Test set early that the Baseline model in the epochs 4 for the Differentiation model and 5 for the One word plus differentiation model, compare to the epoch 6 for the baseline. As a result, our augment model that beat the baseline bring a lower perplexity takes less time to train the models. As the first impression look that these models are a good option to the baseline model.

We found that the models implemented in LSTM structure show the lower perplexity in the Grid search followed by the most consistent results through all the experiments for all sizes of hidden units.
For our augmented model implemented with LSTM, the model One word plus differentiation was the only one that can be the perplexity generate by the Baseline Grove model; the perplexity generates by this model is the lowest perplexity in all the 192 different experiments that we run in our grid search. However, also, the perplexity generate by the Baseline model was the second one in all the experiments, showing how close is the battle to beat this model. Besides the perplexity generated, our model also took less time to be trained and the best epoch founded by the test set happened early in the epoch 6 compare the epoch 7 of the baseline model. In a way, our implementation if improving the baseline model in different aspect with the significance test is needed to corroborate how significant are these results.

To perform our significance test, we took the best of our augment models implemented in the three different cells. As a result, We did not get any model for the RNN structures, GRU achieved general two models and LSTM just one of these models are going to be compared against their equivalence baseline model. We took these models and generated the perplexity value per each word in the test and the validation sets, regarding the significance test, the test consider two steps, the first step is the results in a started phase the second phase is the results when there is an improvement in the data, this test helps us to measure how significant is this improvement between the two steps.

We applied the significance test on our model against the baseline model with a threshold p-value of 0.05 if the value generates by our augment model is lower than this threshold we can reject the null hypothesis and confirm that the results make significant to be considerate as in improvement.
Unfortunately, after performing the test, we got that the results generated by our augment model are not significant and can thus not considered as a reliable improvement to the baseline model, with the closest value generated by the model One word plus differentiation implemented in GRU structure with a p-value of 0.0627 very close to reaching this goal.

We noticed that the dropout does not help in generate models with lower perplexity, we expect that this regularisation technique helps the model to generalise better to on unseen data. A dropout value of zero was proven to provide the best results. We assume that this happens because the size of our Hidden Units is not as big with a value between 200 to 400. We though on applying different regularisation technique as early stopping but since the beginning of the project we define the time for experiments and apply this other regularisation technique will require more time than planned.

We implement all the models using PyTorch Deep Learning Framework. We added comments into the code to make more readable for a reader with computer science knowledge and some deep learning understanding. The code is appropriately sectioned by experiments and results. It is open for future research purposes. The code is attached to this report in a zip file ready to use in Jupyter Notebooks.

7. Conclusion: Alternative Approaches, Future Work and Reflections.

7.1 Simple variants

We found that the models able to generalise the data quicker than we thought, once the model found this optimal value and continue to be trained it affects the probabilities causing the vanishing/exploding gradient in the long sequence generating extremely high perplexity values. We recommend incorporating another regularisation technique in the grid search: Early stopping. Early stopping can help to avoid over-fitting. To incorporate the early stopping, we need to check the Train and Test errors in each epoch; we stop we train phase when then the error in the test set is higher than the previous epoch. Early stopping can contribute to reduce the training time and help to avoid the generation of vanish/exploding gradients

Due to the time limitation of the project, we use the Penn Treebank dataset limited to 10,000 different words. It is a small corpus where the words that are Out of the Vocabulary (OOV) represented by the label <unk>. It would be interesting to train the model on a sample that contains all the different word in the vocabulary without restriction, and that can bring the complete context of the sentences. The Penn Treebank extracted the sentences from a text of the Wall Street Journal (WSJ). In the future, it would be interesting to try other domains to apply this model. One example could be the life-conversations from the text language (Chafe and Tannen, 1987). We propose to use other parameters to perform a more in-depth grid search to find a model that can bring optimal results for the LSTM and GRU respectably to the model that achieve lower perplexity than the baseline.

7.2 Alternative Word Embeddings representation

The embeddings were implemented using the Glove pre-trained word vector representation. However, there are other pre-trained word vectors that can be used to perform a similar task a the well-known word2vec. It is known that the words tokens have beneficent another sophisticated implementation. The word2vector (Mikolov et al., 2013) have shown to give good results on representing semantic and syntactical relationships. One example could be Fast Text (Joulin et al., 2016). The main topic on the use of this other pre-trained vector representation could be to cooperate again Glove to figure out which can grind more significant between the words vectors to create a model that can beat the baseline model with a more significant margin to be able to pass the significance test.

7.3 The Logic Tensor Network

This project is a contribution to the research line of the Dr Tillman Weyde; he is one of the primary beneficiaries. One of the main lines of the main lines of research of the Dr Tillman Weyde is the audio processing. If we can create a model that help on the word prediction it can have any application on different areas, like speech recognition, audio pre-processing, word generations and many more. The Logic Tensor Network (LTN) was suggested to implement a Statistical Language model (SLM) as we mentioned in section 1 but due to time constraints of the project, we decide to update the scope. LTN is developed in Google TensorFlow, the code of our project was developed in PyTorch to make use of our code, it would be necessary to migrate to TensorFlow.

7.4 Reflections

This project was done as a three months dissertation to complete the course of MSc Data Science. One of the main objectives was to help to the Research Centre of Machine Learning at City, University of London to create new areas of investigation to give a different point of view of future PhD of Dissertation projects. It was an exciting and challenging project where we start we a simple idea that was expanded by creating a variant of the models that were being made as a standard for N-Gram models implemented pretrained embeddings vectors.

I did not have previous knowledge of Natural Language Processing (NLP) implementation. NLP is an important topic right now in Deep Learning. With some many areas of research, it was not easy to find a project that can fit the suitable requirements and timeframe.

The initial idea was too ambitious for the timeframe provided to complete a three months project. However, after a literate review and the guideline of our supervisor Dr Tillman Weyde, it was decided to use the distance between the vector presentation as a new idea.

One of the main obstacles was to learn a new deep learning framework that we were not used before in a reduce period, gain expertise and be able to create a model in PyTorch fluently. We started with some tutorial form the PyTorch documentation to get more confidence in the framework before start creating prototypes for the project. Our coding skills in Python also increase, cause the needs to use PyTorch. In a reasonable period, I was able to create models that allow me to answer the research question and create all the necessary analysis to complete the dissertation project successfully. As a result, we develop such a valuable research skill that helps to document correctly the job that is already done and explain properly the way that we are contributing to our work done in this project.

I finish this master's degree with the satisfaction that I was able to complete a novel project in a reduced time frame that involved the definition of the problem, learning a new framework and delivering a high-quality report. I am looking forward to implementing my new skill in the industry and expand my knowledge in Deep Learning.

8. References

Anish Singh, 2017, The Vanishing Gradient Problem, https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257

Artur d'Avila Garcez and and Luciano Serafini, 2016, Logic Tensor Networks

Audhkhasi, K., Sethy, A. & Ramabhadran, B. 2016, "Semantic word embedding neural network language models for automatic speech recognition", IEEE, , pp. 5995.

Beaufais, F., 2015. "The neural networks behind Google Voice transcription."

Bellman, R. E. (1957). Dynamic programming. Princeton, NJ: Princeton University Press

Bengio, Y., 2008. Neural net language models. Scholarpedia, 3(1), p.3881.

Bengio, Y., Ducharme, R., 2001. A neural probabilistic language model. In: Proceedings of Advances in Neural Information Processing Systems, vol. 13. pp. 932–938

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C., 2003. A neural probabilistic language model. Journal of machine learning research, 3(Feb), pp.1137-1155.

Bengio, Y., Ducharme, R., Vincent, P., 2000. A neural probabilistic language model. Technical Report 1178, De´partement d'informatique et recherche ope´rationnelle, Universite´ de Montre´al.

Bengio, Y., Frasconi, P. & Simard, P. 1993, "The problem of learning long-term dependencies in recurrent networks", IEEE, , pp. 1183.

Bengio, Y., Simard, P. & Frasconi, P. 1994, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157-166.

Bhoir, S., Ghorpade, T. & Mane, V. 2017, "Comparative analysis of different word embedding models", IEEE, , pp. 1.

Bird, S., Klein, E. and Loper, E., 2009. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".

Britz, D., 2015. Recurrent neural network tutorial, part 4-implementing a GRU/LSTM RNN with python and theano. URL http://www. wildml. com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano.

Brownlee, J. (2017). A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size - Machine Learning Mastery. Available at: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configurebatch-size/ [Accessed 24 Dec. 2017].

Chafe, W. and Tannen, D., 1987. The relation between written and spoken language. Annual Review of Anthropology, 16(1), pp.383-407

Chelba, C., Norouzi, M. & Bengio, S. 2017, "N-gram Language Modeling using Recurrent Neural Network Estimation", .

Chen, Stanley F. and Joshua Goodman. 1998. An Empirical Study of Smoothing Techniques for Language Modeling. Harvard Computer Science Group Technical Report TR-10-98.

Chomsky, N., 1957. Syntactic Structures. The Hague: Mouton..(1965), Aspects ofthe Theory of Syntax.
Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Church, K.W. 2017, "Word2Vec", Natural Language Engineering, vol. 23, no. 1, pp. 155-162.
Deep Learning @ Home. (2017). Jun 11, 2017 - Simple LSTM based language model in PyTorch vs Tensorflow. [online] Available at: http://deeplearningathome.com/2017/06/PyTorch-vs-Tensorflowlstm-language-model.html [Accessed 23 Jan. 2018].

Dense Embeddings PyTorch, 2018, Getting Dense Word Embeddings, https://pytorch.org/tutorials/beginnerlp/word_embeddings_tutorial.html#getting-dense-word-embeddings

Dey, R. & Salem, F.M. 2017, "Gate-variants of Gated Recurrent Unit (GRU) neural networks", IEEE, , pp. 1597.

Dey, R. and Salem, F.M., 2017. Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks. arXiv preprint arXiv:1701.05923.

Donadello, I., Serafini, L. and Garcez, A.D.A., 2017. Logic Tensor Networks for Semantic Image Interpretation. arXiv preprint arXiv:1705.08968

E. Cambria, S. Poria, A. Gelbukh, and M. Thelwall, 2017, “Sentiment analysis is a big suitcase,” IEEE Intelligent Systems, vol. 32, no. 6, pp. 74–80, 2017.

Elman, J.L. 1990, "Finding structure in time", Cognitive Science, vol. 14, no. 2, pp. 179-211.

Foram S. Panchal , Mahesh Panchal, 2014, "Review on Methods of Selecting Number of Hidden Nodes in Artificial Neural Network, IJCSMC, Vol. 3, Issue. 11, pg.455 – 464"

Gal, Y. & Ghahramani, Z. 2015, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks",

Gers, F.A., Schmidhuber, J. & Cummins, F. 1999, "Learning to forget: continual prediction with LSTM", IEE, London, pp. 850.

Grachev, A.M., Ignatov, D.I. & Savchenko, A.V. 2017, "Neural Networks Compression for Language Modeling", .

Graves, A., Mohamed, A.R. and Hinton, G., 2013, May. Speech recognition with deep recurrent neural networks. In Acoustics, speech and signal processing (icassp), 2013 ieee international conference on (pp. 6645-6649). IEEE.

Gupta, A. & Tripathy, B.K. 2017, "Implementing GloVe for context based k-means++ clustering", IEEE, , pp. 1041.

Hochreiter, S. & Schmidhuber, J. 1997, "Long Short-Term Memory", Neural Computation, vol. 9, no. 8, pp. 1735-1780.

Hole, G. (2011). The Wilcoxon test., Users.sussex.ac.uk: "http://users.sussex.ac.uk/~grahamh/RM1web/WilcoxonHandoout2011.pdf"

Hu, Y., Huber, A., Anumula, J. & Liu, S. 2018, "Overcoming the vanishing gradient problem in plain recurrent networks",

Inan, H., Khosravi, K. & Socher, R. 2016, "Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling", .

J. L. Elman, 1991, "Distributed representations, simple recurrent networks, and grammatical structure", Machine learning, vol. 7, no. 2–3, pp. 195–225.

J. Weston, S. Bengio, and N. Usunier, 2011, "Wsabie: Scaling up to large vocabulary image annotation," in IJCAI, vol. 11, 2011, pp. 2764–2770.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation

Joris Pelemans, Noam Shazeer, and Ciprian Chelba, 2016 Sparse non-negativematrix languagemodeling. Transactions of the Association for Computational Linguistics, 4:329–342, ISSN 2307-387X

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H. and Mikolov, T., 2016. FastText. zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N. & Wu, Y. 2016, "Exploring the Limits of Language Modeling", .

Jun-Li, W., Xiao-Min, W. & Ya-Xing, Y. 2017, "A word embedding generation method based on flexible length context", IEEE, , pp. 2564.

Kaiser, Ł. & Sutskever, I. 2015, "Neural GPUs Learn Algorithms", .

Kamkarhaghighi, M. & Makrehchi, M. 2017, "Content Tree Word Embedding for document representation", Expert Systems With Applications, vol. 90, pp. 241-249.

Krishnamurthy, B., Puri, N. & Goel, R. 2016, "Learning Vector-space Representations of Items for Recommendations Using Word Embedding Models", Procedia Computer Science, vol. 80, pp. 2205-2210.

Levy, O., & Goldberg, Y. 2014. "Dependency-Based Word Embeddings. ACL."

Li, M., Zhang, T., Chen, Y. & Smola, A. 2014, "Efficient mini-batch training for stochastic optimization", ACM, , pp. 661.

Li, Z., Kulhanek, R., Wang, S., Zhao, Y. & Wu, S. 2017, "Slim Embedding Layers for Recurrent Neural Language Models", .

Lipton, Z.C., Berkowitz, J. & Elkan, C. 2015, "A Critical Review of Recurrent Neural Networks for Sequence Learning", .

Makarenkov, V., Shapira, B. & Rokach, L. 2016, "Language Models with Pre-Trained (GloVe) Word Embeddings".

Marcus, M.P., Marcinkiewicz, M.A. and Santorini, B., 1993. Building a large annotated corpus of English: The Penn Treebank. Computational linguistics, 19(2), pp.313-330.

Marcus, Mitchell, et al. 1999, "Penn Treebank-3 LDC99T42. Web Download. Philadelphia: Linguistic Data Consortium"

Martín Pellarolo 2018, "How to use Pre-trained Word Embeddings in PyTorch" https://medium.com/@martinpella/how-to-use-pre-trained-word-embeddings-in-pytorch-71ca59249f76

Martin Sundermeyer, Ralf Schluter, and Hermann Ney, 2012, "LSTM Neural Networks for Language Modeling"

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

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J. and Khudanpur, S., 2010 "Recurrent neural network based language model. In Interspeech (Vol. 2, p. 3)."

Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastasopoulos, A., Ballesteros, M., Chiang, D., Clothiaux, D., Cohn, T., Duh, K., Faruqui, M., Gan, C., Garrette, D., Ji, Y., Kong, L., Kuncoro, A., Kumar, G., Malaviya, C., Michel, P., Oda, Y., Richardson, M., Saphra, N., Swayamdipta, S. & Yin, P. 2017, "DyNet: The Dynamic Neural Network Toolkit",

N-Grams Pytorch, 2018, "An Example: N-Gram Language Modeling, https://pytorch.org/tutorials/beginnerlp/word_embeddings_tutorial.html#an-example-n-gram-language-modeling"

Olah, C., 2015. "Understanding LSTM Networks--colah's blog. Colah"

Oualil, Y., Greenberg, C., Singh, M. & Klakow, D. 2017, "Sequential Recurrent Neural Networks for Language Modeling",.

Oualil, Y., Singh, M., Greenberg, C. & Klakow, D. 2017, "Long-Short Range Context Neural Networks for Language Modeling", .

P. D. Turney and P. Pantel, 2010, “From frequency to meaning: Vector space models of semantics,” Journal of artificial intelligence research, vol. 37, pp. 141–188, 2010.

Panchal, G., Ganatra, A., Kosta, Y.P. and Panchal, D., 2011. "Behaviour analysis of multilayer perceptronswith multiple hidden neurons and hidden layers. International Journal of Computer Theory and Engineering, 3(2), p.332."

Paulus, R., Socher, R. and Manning, C.D., 2014. Global belief recursive neural networks. In Advances in Neural Information Processing Systems (pp. 2888-2896).

Press, O. & Wolf, L. 2016, "Using the Output Embedding to Improve Language Models", .

R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, 2011 “Parsing natural scenes and natural language with recursive neural networks,” in Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 129–136.

Robert C. Moore , Chris Quirk "Less is More: Significance-Based N-gram Selection for Smaller, Better Language Models"

Rohan Kapur, 2016, Rohan "#4: The vanishing gradient problem Oh no — an obstacle to deep learning!, https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b"

Rosenfeld, R., 2000. Two decades of statistical language modeling: Where do we go from here?. Proceedings of the IEEE, 88(8), pp.1270-1278.

Ruder, S. 2016, "An overview of gradient descent optimization algorithms", .
Scipy, scipy.stats.wilcoxon, 2014, "https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.wilcoxon.html"

Serafini, L. and Garcez, A.D.A., 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge.

Shi, T. & Liu, Z. 2014, "Linking GloVe with word2vec", .

Shi, Y., Wiggers, P. and Jonker, C.M., 2012. Towards recurrent neural networks language models with linguistic and contextual features. In Thirteenth Annual Conference of the International Speech Communication Association.

Socher, Richard; Bauer, John; Manning, Christopher; Ng, Andrew, 2013, "Parsing with compositional vector grammars"

Socher, Richard; Perelygin, Alex; Wu, Jean; Chuang, Jason; Manning, Chris; Ng, Andrew; Potts, Chris, 2013, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank"
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. 2014, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JOURNAL OF MACHINE LEARNING RESEARCH, vol. 15, pp. 1929-1958.

Taranpreet Singh Rai, 2017. Using Structural Information in Neural Language Models

Verwimp, L., Pelemans, J., Van hamme, H. & Wambacq, P. 2017, "Character-Word LSTM Language Models", .
Wallace Chafe, and Deborah Tannen 1987, "The Relation Between Written and Spoken Language".

Williams, R.J. and Peng, J., 1990. "An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural computation, 2(4), pp.490-501"
Woolson, R.F., 2008. "Wilcoxon Signed-Rank Test. Wiley encyclopedia of clinical trials"

Word Embeddings PyTorch, 2018, "Word Embeddings: Encoding Lexical Semantics, https://pytorch.org/tutorials/beginnerlp/word_embeddings_tutorial.html#word-embeddings-encoding-lexical-semantics"

Yin, W., Kann, K., Yu, M. & Schütze, H. 2017, "Comparative Study of CNN and RNN for Natural Language Processing".

Young, T., Hazarika, D., Poria, S. & Cambria, E. 2017, "Recent Trends in Deep Learning Based Natural Language Processing".

Yu, D.J., 2016. Part-Of-Speech Tag Embedding for Modeling Sentences and Documents.

Zamani, H. & Croft, W. 2016, "Embedding-based Query Language Models", ACM, pp. 147.

Zaremba, W., Sutskever, I. & Vinyals, O. 2014, "Recurrent Neural Network Regularization"

Appendix A – Original Proposal

Research Project Proposal:

Neural Statistical Language model using Logic Tensor Networks

Francisco Javier Medel Medina
MSc Data Science

Supervisor: **Dr Tillman Weyde**

April 2018

1. Purpose

1.1 Introduction

Natural Language Processing (NLP) is a field in Machine Learning, which has still room for improvement. The idea is to use Statistical Relational Learning (SRL) to improve the results of predictions. We plan to use the Logic Tensor Networks (LTN) framework for a Neural Network architecture in Statistical Language model, i.e., a model that predicts the next token in a given context. One way to improve Neural Network Language models is to add knowledge about the grammatical rules that exist in the English Language to the model. We will use grammar rules as Background Knowledge (BK) to build a model using Logic Tensor Networks (LTN), to improve the results of Neural Statistical Language models.

1.2 Research question

Can the use of grammatical rules as Background Knowledge be applied in the Language modelling Techniques to increase the accuracy of predicted phases in a model-built using Logic Tensor Network framework?

1.3 Objectives

- Statistical Relational Learning (SLR)
    - Create grammatical rules as input for Background Knowledge (BK)
    - Improve existing Neural Statistical Language models with Background Knowledge (BK) on Logic Tensor Network (LTN) framework.
- Explore existing Techniques and compare results.
    - Try other approaches like n-grams (unigrams, bigrams, trigrams) and Bag Of Words (BOW) that can predict text and compare the result with the Model generate in Logic Tensor Network.

1.4 Beneficiaries and Project Impact

The purpose of this project is to contribute to the development of Neural statistical Language models under the supervision of Dr Tillman Weyde using the Logic Tensor Network (LTN) framework developed by the Research Centre of Machine Learning at City, University of London (Artur d'Avila Garcez and Luciano Serafini 2016). We will benefit from this project as we aim to obtain skills and knowledge in one of the most exciting areas of Machine Learning: Natural Language Processing. We believe by contributing to this project; We will contribute to this field. The implementation of a model on the Logic Tensor Network with this new approach, it can contribute to Dr Weyde's research for future projects bringing valuable results, that can help to improve the model and develop new research areas for Postgraduate and PhD Students. The final milestone is to evaluate the accuracy of the text generation model in automatic translation application.

2. Critical Context

2.1 Statistical Language models

The purpose of a Statistical Language model or Language modelling is to determine a probability distribution for the next word in a given context. The occurrence of words learns the probability in examples of text, often huge corpus.

These days, the use of Neural Networks has become widespread in solving machine learning problems. The use of Neural Networks with Statistical Language modelling is called Neural Statistical Language modelling. This approach has shown better results than traditional methods like n-grams or bag-of-words, these methods are relatively simple to train, but they sometimes show poor result because of the curse of dimensionality. The curse of dimensionality is a phenomenon that is so common in NLP for a higher number of word combinations.

Bengio et al. (2003), implemented Neural Language models (NLM) where the words are converted to a vector to be parameterised, this process is known as word embeddings. These word embeddings are then used in the inputs layer of a neural network. This helps overcome the curse of dimensionality because the word embedding helps to reduce the number of inputs in the Neural Network.

2.2 Syntax in the Language

The syntax in a language can be described in the form of grammar. The syntax describes the order of the words in a sentence, and how the words are used together. In a sense, the syntax is a kind of validator that is the sentences are structured correctly. For example, the English language mostly follows an SVN (Subject, Verb, Object) structure.
I (Subject) + love (Verb) + travelling (Object)

However, this basic structure is not enough to express complex sentences. Some grammar studies Huddleston, R. (1988) specify that the English Language has between 8 and 12 parts. For this project, we are going to focus on the next nine syntactic categories as *Verb, Noun, Adjective, Determiner, Adverb, Pronoun, Preposition*, *Conjunction* and *Interjection* Baker, M. (2003).

Consider the following sentence:

interjection	pron.	conj.	det.	adj.	noun	verb	prep.	noun	adverb
Well,	she	and	my	young	sister	walk	to	park	slowly.

Our primary hypothesis is that extracting these rules as Background Knowledge and converting to real logic to feed into the Neural Network using in the Logic Tensor Network framework will bring a more reliable model that is more consistent on dealing with noisy data and good on quality and performance.

To create the model, a set of suitable the grammatical rules needs to be identified, which will describe the most common sentence structures and will have to be conditional on sentence types. These rules will be implemented in the layers of the Neural Network model to activate the neurons for those phrases that are grammatically correct. The purpose is to provide the neural networks with the Background Knowledge (BK) rules as a starting point, where there is prior knowledge of English Language structure, and thus less learning from data is required. We believe that this implementation would also give better results on noisy data.

One factor to consider is, even where there are grammatical rules for the language that the natural language often does not commit them. The natural language has a vast variety of terms and combination that introduce ambiguities to the sentences but the language and can be still understood by humans. This will be addressed by conditioning the structures on sentence types, such that the structures would not be applied if the sentence does not have to a grammatical structure.

2.3 Logic Tensor Networks (LTN) and Statistical Relational Learning (SRL)

Statistical Relational Learning (SRL) is a field of Artificial Intelligence (AI) and more particularly a subdiscipline of Machine Learning that deals with relational learning context in complex structures using statistical learning methods. SRL usually uses the first order logic to describe relation properties and tries to describe formalisms in a general manner (Ravkic et al., 2015).

To deal with the uncertainty that is prevalent in a real-world problem, logical and relational representation can be expressed by probability theory. This opens the door to the field of statistical relational learning (SRL) (Getoor and Taskar, 2007), that is also known as Probabilistic Logic Learning (De Raedt and Kersting, 2003) that deals with probabilistic approaches to Logical Representation. In Figure 1. We can see how Statistical Relational Learning (SRL) of Natural Language is a research area of Computational Natural Language Learning that emerges at an intersection between Natural Language Processing (NLP) and Machines Learning (ML).

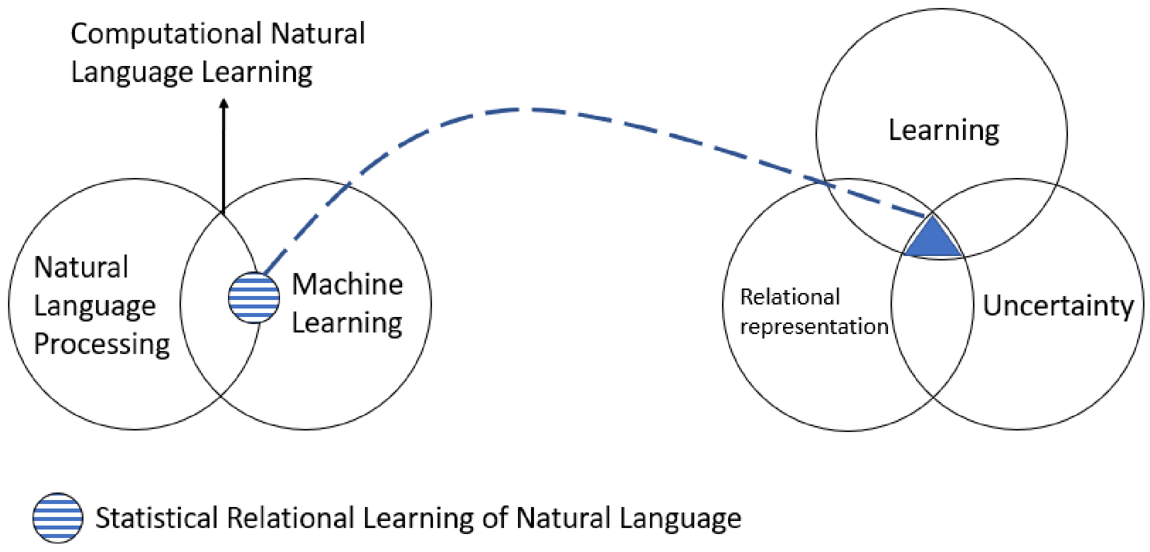

Computational Natural Language Learning
Natural Language Processing
Machine Learning
Learning
Relational representation
Uncertainty
Statistical Relational Learning of Natural Language

Figure. 1 Situating statistical relational learning of natural language (Bianchini, Maggini, 2013)

Logic Tensor Networks (LTN) is a framework that helps build a neural network that must deal with Statistical Relational Learning by applying Real Logic (Serafini and Garcez, 2016). One of the critical factors for this project is to find suitable grammatical rules as a Background Knowledge that represent the syntax of the English language to the Neural Network that allow us to implement the Logic Tensor Network to create a model that can predict the words in a phrase based on both the given background knowledge and learning from data.

3. Approach

3.1 Software and Frameworks

Python 3.5 will be the primary language to develop this project. The main libraries to use will be NumPy, Pandas and Scikit-Learn. For the framework, I will use Google TensorFlow in the version 1.7.0. Data pre-processing techniques will be implemented to prepare the data for implementation.

3.2 Data sources and Pre-processing.

Datasets of English texts will be used for the project. The English language has its structure, which we aim to make use of by applying grammatical rules. We are going to use a popular dataset, the Stanford Natural Language Inference (SNLI) Corpus Samuel R. Bowman (2015). The dataset is a collection of five hundred and seventy thousand English sentences. The sentences are labelled in different categories as contradiction, natural and entailment. The grammatical structure sentences are the source for the background knowledge rules. We are going to do the pre-processing with The Natural Language Toolkit (NLTK). The token is going to be taken from these categories. NLTK has complete documentation (Bird, Steven, 2009) that focuses on tagging and analysing sentence structures; this pre-processing is going to be fundamental to get data ready to feed into deep neural networks.

3.3 Early Investigation

To get more experience on this filed of NPL that can help to complete this project, We plan to complete a set of tutorials related to the construction of NLP and the previous steps that the process demand. WE believe this will help us understand the Language modelling better.

In Language modelling, there are other exciting approaches like Long Short-Term Memory (LSTM) (Hoch Reiter and Schmid Huber, 1997; Gers et al., 1999). One of the applications is how the LSTM is applied to infer the word meaning in large pieces of text and how can fix with other techniques like contextual features. As a result, we can get text generation, word prediction or topic prediction (Le, P., Dymetman, M. & Renders, J. 2016).

I also plan to explore some other more complicated techniques. A key factor for a successful project is to develop proper grammatical rules to feed the LTN. NLP is a mature research field, and other useful techniques exist to generate BK. Therefore, it will be good to document if I find any another interesting approach that can this project.

3.4 Framework Implementation

Logistic Tensor Network is a framework that uses Google's TensorFlow (Abadi 2016). LTN uses tensor networks combined with the first-order logic that enable the background knowledge. To understand first-order logic and the mathematical formulation of Real Logic is crucial to working with the LTN framework. I plan to complete a tutorial in TensorFlow about Neural Statistical Language models to get more ability on the implementation of the LTN. The grammatical rules will be determined in the pre-processing state using NLTK. These grammatical rules will then the implemented in the LTN. The idea is to integrate the use of LTN with Recurrent Neural Networks (RNN) to improve the results in Language modelling area.

3.4.1 Model Evaluation

One useful technique to evaluate Statistical Language models is *perplexity* (Martin and Jurafsky, 2014) that is based on the cross-entropy between the data and the prediction. The calculation is made over the probability distribution of words (or phrases). As a result, a good model is one that gives high probabilities score, and thus a low perplexity, on the test data (J. Hockenmaier 2015).

4. Work plan

In this section, I describe the work plan of the proposal as shown in the Figure 2 Gantt chart. The project life runs from July 1st until October 1st, 2018. We find four milestones until the dead project line. The first milestone is completing the literature; the second one is to prepare the dataset, the third one is to correctly setup environment. The fourth and final milestone is to complete the project documentation and hand over the project. If there are some external or sudden changes, the plan will be updated to show these new changes.

Task Name	July					Aug					Sep				Oct
	1	6	13	20	27	3	10	17	24	31	7	14	21	28	1
Literature review	🟦														
Literature reserch	🟦														
Suppervisor Meeting		🟦													
Data gattering		🟦													
Write Proposal			🟦	🟦	🟦	🟩	🟩	🟩	🟩	🟩	🟨	🟨	🟨	🟨	
Suppervisor Metting			🟦												
Explore Rules for Logic Tensor Networks			🟦	🟦											
Get expersice in the Coding Implentation				🟦	🟦										
Get expersice in Logic Tensor Networks				🟦	🟦										
Start Tensor Tutorial					🟦										
Brake						🟩									
Write Litterature Review							🟩	🟩							
Start writing first draft dissertation								🟩	🟩						
Data Preprocess with NLTK and other passers								🟩							
Suppervisor Metting									🟩	🟩					
Implement and Test LTN with Language Models										🟩					
Evaluate Model										🟩	🟨				
Implement and Test LTN with various rules applied											🟨				
Evaluate Model											🟨	🟨			
Write Methodology and dissertation												🟨			
Write results anc critical evaluation												🟨			
Commented code and update appendices													🟨		
Complete dissertation first draft														🟨	
Final report submition															🟥

Figure 2 Work Plan Schedule

5. Risks

How to manage and to mitigate possible risks that can affect the project is a key component. I found Dawson's (2006) framework for risk management very useful in this context. A list of possible risk that can affect the project success has been observed.

I have a list of possible risks, outlined by nature. Each risk has assigned a likelihood from 1 to 5, Each risk consequence scale from 1 to 5, I calculate the impact by (likelihood * consequence). For each risk, I have mitigating steps to reduce the likelihood that the risk happened, and I have to a contingency plan if the risk eventuates.

#	Description	Likelihood	Consequence	Impact	Mitigation	Contingency
1	Difficulties with coding in TENSORFLOWTM	3	3	6	Spend least four weeks on tutorial and complete some mini projects	Ask the supervisor to help overcome this gap.
2	Difficulty in understanding the Language model techniques	3	4	12	Select carefully literature that help to understand the models.	Take an extra course that can help to get some practical knowledge
3	LTN does not produce the expected results	3	4	12	In a research project it is possibility. Keep posted the supervisor of realistic results.	Document well all the effort, emphasize the future work for future projects.
4	Tasks taking longer that the first schedule	3	5	15	Carefully check the project plan and the time assigned to each task.	Move task with high uncertainty to fist period of the project.
5	Laptop fails/stolen.	1	5	5	Take extra care of always have backup on the cloud of the last version	Purchase a new Laptop
6	Personal computer cannot process computational task and it takes longer	2	2	4	Set up the VPN to access the servers of the University. And setup an environment for the project.	Use the server that the university give to train the models
7	Loss of interest	1	5	5	The topic was carefully chosen	Talk with the supervisor and check the state of the art of new application to get a extra motivation.

6. Ethics Review Form

This project only involves technical and coding work, most of the task is apply statistical models and machine learning models on the computer of the author. This project does not work on data that can help to identify persons individually.

Ethics Review Form: BSc, MSc and MA Projects
Computer Science Research Ethics Committee (CSREC)

Undergraduate and postgraduate students undertaking their final project in the Department of Computer Science are required to consider the ethics of their project work and to ensure that it complies with research ethics guidelines. In some cases, a project will need approval from an ethics committee before it can proceed. Usually, but not always, this will be because the student is involving other people ("participants") in the project.
In order to ensure that appropriate consideration is given to ethical issues, all students must complete this form and attach it to their project proposal document. There are two parts:
Part A: Ethics Checklist. All students must complete this part. The checklist identifies whether the project requires ethical approval and, if so, where to apply for approval.
Part B: Ethics Proportionate Review Form. Students who have answered "no" to questions 1 – 18 and "yes" to question 19 in the ethics checklist must complete this part. The project supervisor has delegated authority to provide approval in this case. The approval may be provisional: the student may need to seek additional approval from the supervisor as the project progresses.

A.1 If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval.		*Delete as appropriate*
1.	Does your project require approval from the National Research Ethics Service (NRES)? For example, because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/.	**No**
2.	Does your project involve participants who are covered by the Mental Capacity Act? If so, you will need approval from an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/.	**No**
3.	Does your project involve participants who are currently under the auspices of the Criminal Justice System? For example, but not limited to, people on remand, prisoners and those on probation? If so, you will need approval from the ethics approval system of the National Offender Management Service.	**No**

A.2 If your answer to any of the following questions (4 – 11) is YES, you must apply to the City University Senate Research Ethics Committee (SREC) for approval (unless you are applying to an external ethics committee).		*Delete as appropriate*
4.	Does your project involve participants who are unable to give informed consent? For example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	**No**
5.	Is there a risk that your project might lead to disclosures from participants concerning their involvement in illegal activities?	**No**
6.	Is there a risk that obscene and or illegal material may need to be accessed for your project (including online content and other material)?	**No**
7.	Does your project involve participants disclosing information about sensitive subjects? For example, but not limited to, health status, sexual behaviour, political behaviour, domestic violence.	**No**
8.	Does your project involve you travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (See http://www.fco.gov.uk/en/)	**No**

CITY

9.	Does your project involve physically invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	**No**
10.	Does your project involve animals?	**No**
11.	Does your project involve the administration of drugs, placebos or other substances to study participants?	**No**

A.3 If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		*Delete as appropriate*
12.	Does your project involve participants who are under the age of 18?	**No**
13.	Does your project involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	**No**
14.	Does your project involve participants who are recruited because they are staff or students of City University London? For example, students studying on a specific course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	**No**
15.	Does your project involve intentional deception of participants?	**No**
16.	Does your project involve participants taking part without their informed consent?	**No**
17.	Does your project pose a risk to participants or other individuals greater than that in normal working life?	**No**
18.	Does your project pose a risk to you, the researcher, greater than that in normal working life?	**No**

A.4 If your answer to the following question (19) is YES and your answer to all questions 1 – 18 is NO, you must complete part B of this form.		
19.	Does your project involve human participants or their identifiable personal data? For example, as interviewees, respondents to a survey or participants in testing.	**No**

7. References:

The Stanford Natural Language Inference (SNLI) Corpus **https://nlp.stanford.edu/projects/snli/**

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M. and Ghemawat, S., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint
arXiv:1603.04467.

Artur d'Avila Garcez and and Luciano Serafini, 2016, Logic Tensor Networks

Bengio, Y., Ducharme, R., Vincent, P. and Jauvin, C., 2003. A neural probabilistic Language model. Journal of machine learning research, 3(Feb)

Baker, M. (2003). Lexical Categories: Verbs, Nouns and Adjectives (Cambridge Studies in Linguistics). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511615047

De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In Shai Ben-David, John Case, and Akira Maruoka, editors, ALT, volume 3244 of Lecture Notes in Computer Science, pages 19–36. Springer, 2004. ISBN 3-540-23356-3.

Daniel Jurafsky and James H. Martin, 2000,"Speech and Language Processing"

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation

Getoor and Taskar, 2007, Introduction to Statistical Relational Learning. MIT Press

Kim, Y., Jernite, Y., Sontag, D. and Rush, A.M., 2016, February. Character-Aware Neural Language models. In AAAI

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)

Jurafsky, D. and Martin, J.H., 2014. Speech and language processing (Vol. 3). London: Pearson.

Huddleston, R. (1988). English Grammar: An Outline. Cambridge: Cambridge University Press. doi:10.1017/CBO9781139166003
Bird, Steven, Edward Loper and Ewan Klein ,2009, *Natural Language Processing with Python*. O'Reilly Media Inc.

J. Hockenmaier, 2015, CS447: Natural Language Processing

Bianchini, M., Maggini, M., Jain, L.C. & SpringerLink eBook Collection, 2013, Handbook on Neural Information Processing, Springer Berlin Heidelberg, Berlin, Heidelberg.

Le, P., Dymetman, M. & Renders, J. 2016, "LSTM-based Mixture-of-Experts for Knowledge-Aware Dialogues".

Appendix B – Preliminary Grid Search Results

RNN - Baseline Glove Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	85	300	400	0	53.96	6.806	15	53.396	3
2nd	37	300	200	0	43.23	12.007	15	54.13	3
3rd	13	100	200	0	42.48	13.519	15	55.176	4
4th	169	200	400	0.2	81.15	13.248	15	55.503	3
5th	25	200	200	0	46.25	13.39	15	55.637	4
6th	133	300	200	0.2	47.17	21.077	15	56.239	3
7th	61	100	400	0	78.85	8.692	15	56.268	3
8th	49	50	400	0	78.36	9.382	14	56.604	5
9th	73	200	400	0	80.55	7.364	15	56.811	3
10th	109	100	200	0.2	29.4	23.456	15	56.859	5
11th	121	200	200	0.2	29.63	22.462	15	57.264	5
12th	1	50	200	0	38.09	15.603	15	57.512	4
13th	181	300	400	0.2	76.33	12.435	15	57.903	5
14th	157	100	400	0.2	80.42	14.304	15	58.949	2
15th	145	50	400	0.2	71.91	15.197	15	59.426	4
16th	97	50	200	0.2	35.92	24.823	15	61.887	6

RNN – Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	88	300	400	0	79	7.726	14	54.642	3
2nd	40	300	200	0	29.48	13.178	14	55.775	4
3rd	76	200	400	0	53.74	7.622	15	55.817	3
4th	64	100	400	0	80.52	8.225	15	56.43	4
5th	16	100	200	0	43.31	14.268	15	56.521	3
6th	172	200	400	0.2	81.03	14.241	14	56.948	5
7th	28	200	200	0	47.99	13.501	15	57.034	2
8th	160	100	400	0.2	80.79	14.571	15	57.102	3
9th	52	50	400	0	78.56	9.239	15	57.152	3
10th	184	300	400	0.2	164.6	13.736	14	57.965	3
11th	136	300	200	0.2	48.49	22.376	14	58.114	3
12th	124	200	200	0.2	29.32	22.66	15	58.159	3
13th	4	50	200	0	41.57	16.004	15	58.843	3
14th	112	100	200	0.2	29.46	23.994	14	59.13	3
15th	100	50	200	0.2	28.74	26.763	13	59.621	4
16th	148	50	400	0.2	79.87	15.896	15	61.955	2

RNN – One-word plus Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	43	300	200	0	28.56	13.5689	13	54.7393	3
2nd	175	200	400	0.2	171.56	13.6184	15	55.4471	3
3rd	91	300	400	0	53.96	7.5035	15	55.7516	2
4th	31	200	200	0	46.22	14.0701	14	56.4347	3
5th	67	100	400	0	78.03	8.2963	15	56.4388	4
6th	187	300	400	0.2	148.34	13.3489	15	56.6999	5
7th	19	100	200	0	43.26	14.4397	15	57.0223	3
8th	55	50	400	0	78.18	9.2725	15	58.1079	4
9th	79	200	400	0	50.06	7.854	15	58.2917	3
10th	7	50	200	0	38.03	16.4742	14	58.3085	4
11th	115	100	200	0.2	45.33	23.9967	15	58.4682	5
12th	139	300	200	0.2	85.28	20.8701	15	58.5501	4
13th	127	200	200	0.2	29.36	22.3874	15	58.5513	5
14th	163	100	400	0.2	48.53	15.0116	15	59.0174	2
15th	151	50	400	0.2	78.7	15.7353	15	60.3884	4
16th	103	50	200	0.2	35.96	25.2319	15	60.6197	3

RNN – Differentiation Only Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	94	300	400	0	51.46	7.259	15	57.345	3
2nd	46	300	200	0	33.54	13.156	15	58.114	4
3rd	34	200	200	0	45.17	14.684	15	58.414	3
4th	190	300	400	0.2	77.74	13.454	15	58.613	2
5th	82	200	400	0	78.62	8.183	15	58.645	3
6th	178	200	400	0.2	166.1	14.285	14	58.88	2
7th	22	100	200	0	38.44	14.893	15	59.456	3
8th	130	200	200	0.2	45.17	24.299	14	60.067	3
9th	142	300	200	0.2	27.32	22.495	15	60.142	4
10th	166	100	400	0.2	76.82	14.089	15	61.496	4
11th	70	100	400	0	79.33	8.551	14	61.731	5
12th	58	50	400	0	69.73	9.825	15	62.316	3
13th	118	100	200	0.2	36.17	25.247	14	62.66	3
14th	154	50	400	0.2	70.78	15.572	15	63.369	4
15th	10	50	200	0	39.25	16.699	15	63.501	4
16th	106	50	200	0.2	33.94	27.332	14	66.927	5

GRU - Baseline Glove Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	39	300	200	0	35.04	12.671	15	52.254	7
2nd	27	200	200	0	57.7	15.198	15	52.515	7
3rd	135	300	200	0.2	55.67	20.565	15	53.435	7
4th	123	200	200	0.2	56.25	22.103	15	53.611	9
5th	15	100	200	0	50.4	17.413	15	53.788	7
6th	75	200	400	0	93.27	8.214	15	54.226	5
7th	87	300	400	0	91.5	6.885	15	54.314	4
8th	183	300	400	0.2	177.6	11.934	15	54.449	4
9th	111	100	200	0.2	33.72	24.759	15	55.074	9
10th	171	200	400	0.2	138.9	13.621	15	55.184	7
11th	159	100	400	0.2	128.8	15.667	15	55.254	7
12th	63	100	400	0	129.4	9.967	15	55.609	6
13th	3	50	200	0	50.51	20.341	15	55.891	9
14th	99	50	200	0.2	33.34	27.662	15	56.671	12
15th	51	50	400	0	128.7	11.049	15	57.101	8
16th	147	50	400	0.2	141	17.233	15	57.874	7

GRU – Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	42	300	200	0	34.36	11.595	15	52.043	4
2nd	186	300	400	0.2	198.9	11.056	15	53.361	6
3rd	30	200	200	0	57.9	13.848	15	53.445	6
4th	138	300	200	0.2	57.54	19.184	15	53.531	6
5th	78	200	400	0	91.35	6.811	15	53.73	5
6th	18	100	200	0	51.01	15.627	15	53.739	5
7th	174	200	400	0.2	139.7	12.323	15	53.799	5
8th	90	300	400	0	94.63	5.864	15	53.832	5
9th	126	200	200	0.2	60.47	21.477	15	53.854	7
10th	66	100	400	0	130.4	8.45	15	54.124	5
11th	114	100	200	0.2	54.72	24.522	15	54.501	7
12th	162	100	400	0.2	142.9	13.916	15	55.433	7
13th	6	50	200	0	50.5	19.04	15	55.46	8
14th	54	50	400	0	129.3	9.435	15	56.409	4
15th	102	50	200	0.2	33.17	26.444	15	56.48	9
16th	150	50	400	0.2	140.9	15.45	15	57.204	6

GRU – One-word plus Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	45	300	200	0	32.9	12.196	15	52.195	6
2nd	33	200	200	0	56.17	13.781	15	52.332	5
3rd	141	300	200	0.2	111.6	19.707	15	52.603	7
4th	93	300	400	0	91.51	6.75	15	52.654	4
5th	129	200	200	0.2	45.68	21.484	15	52.901	7
6th	189	300	400	0.2	136.3	11.842	15	53.092	6
7th	177	200	400	0.2	137.7	12.99	15	53.456	6
8th	21	100	200	0	51.67	15.81	15	53.652	5
9th	81	200	400	0	89.14	7.168	15	53.735	6
10th	165	100	400	0.2	194.9	14.105	15	54.792	5
11th	69	100	400	0	128.9	8.693	15	54.876	5
12th	117	100	200	0.2	55.51	24.165	15	55.013	9
13th	9	50	200	0	49.94	18.488	15	55.249	7
14th	105	50	200	0.2	33.05	26.098	15	56.812	9
15th	153	50	400	0.2	130.6	15.698	15	57.056	5
16th	57	50	400	0	129	9.817	15	57.175	7

GRU – Differentiation Only Language model

Posi-tion	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	96	300	400	0	88.88	7.273	15	55.9	5
2nd	48	300	200	0	32.56	13.789	15	56.052	6
3rd	36	200	200	0	56.4	15.126	15	56.309	6
4th	144	300	200	0.2	32.09	21.788	15	56.497	7
5th	192	300	400	0.2	131	12.358	15	57.184	4
6th	132	200	200	0.2	32.73	23.529	15	57.681	6
7th	180	200	400	0.2	136.4	13.795	15	57.705	6
8th	24	100	200	0	50.04	17.194	15	58.06	7
9th	84	200	400	0	129	7.695	15	58.068	5
10th	72	100	400	0	128.6	8.95	15	59.178	5
11th	120	100	200	0.2	30.75	25.61	15	59.416	6
12th	168	100	400	0.2	136.4	15.146	15	59.798	6
13th	60	50	400	0	127.5	10.407	15	61.522	5
14th	12	50	200	0	50.35	19.943	15	61.529	6
15th	108	50	200	0.2	32.62	27.866	15	63.404	8
16th	156	50	400	0.2	129.5	15.779	15	64.444	5

LSTM - Baseline Glove Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	38	300	200	0	65.04	19.686	15	51.724	8
2nd	86	300	400	0	97.09	14.249	15	51.847	8
3rd	182	300	400	0.2	180.9	20.528	15	52.153	9
4th	134	300	200	0.2	58.82	27.24	15	52.277	11
5th	122	200	200	0.2	57.12	29.365	15	52.577	11
6th	26	200	200	0	59.37	23.761	15	52.735	9
7th	14	100	200	0	53.07	27.184	15	52.855	13
8th	74	200	400	0	138.3	17.356	15	52.946	10
9th	170	200	400	0.2	137.8	22.891	15	53.331	11
10th	62	100	400	0	136	20.216	15	53.945	8
11th	110	100	200	0.2	34.09	34.714	15	54.016	14
12th	2	50	200	0	51.4	30.507	15	54.569	11
13th	158	100	400	0.2	134.7	26.47	15	54.915	10
14th	50	50	400	0	136.1	22.576	15	55.048	8
15th	98	50	200	0.2	34.05	37.168	15	56.152	15
16th	146	50	400	0.2	139.3	29.472	15	56.308	12

LSTM – Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	185	300	400	0.2	237.1	16.822	15	52.087	7
2nd	89	300	400	0	101.7	10.578	15	52.154	7
3rd	77	200	400	0	101.4	13.276	15	52.318	8
4th	29	200	200	0	61.16	19.737	15	52.667	7
5th	41	300	200	0	36.03	16.858	15	52.694	8
6th	137	300	200	0.2	59.83	24.793	15	52.729	9
7th	173	200	400	0.2	143.4	19.287	15	52.776	7
8th	17	100	200	0	53.07	23.077	15	53.068	9
9th	125	200	200	0.2	62.62	27.278	15	53.264	11
10th	65	100	400	0	138	16.502	15	53.586	7
11th	113	100	200	0.2	57.13	31.155	15	53.849	13
12th	5	50	200	0	52.65	28.426	15	54.301	12
13th	53	50	400	0	136.6	18.214	15	54.623	10
14th	161	100	400	0.2	138.2	22.819	15	55.02	9
15th	101	50	200	0.2	34.8	35.805	15	55.486	14
16th	149	50	400	0.2	136.4	26.474	15	56.248	12

LSTM – One-word plus Differentiation Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	44	300	200	0	34.25	17.805	15	51.347	6
2nd	92	300	400	0	97.42	11.707	15	51.43	8
3rd	32	200	200	0	58.86	20.215	15	51.926	10
4th	140	300	200	0.2	127	25.145	15	51.956	10
5th	188	300	400	0.2	236.5	17.986	15	51.992	7
6th	128	200	200	0.2	34.94	27.515	15	52.353	11
7th	80	200	400	0	94.33	14.223	15	52.464	7
8th	20	100	200	0	53.91	23.082	15	52.598	9
9th	176	200	400	0.2	114.7	20.359	15	52.606	9
10th	68	100	400	0	136.9	16.538	15	52.647	8
11th	116	100	200	0.2	58.52	31.243	15	53.616	13
12th	164	100	400	0.2	94.32	22.899	15	53.753	10
13th	8	50	200	0	52.73	27.621	15	54.165	12
14th	56	50	400	0	136	19.87	15	54.307	9
15th	104	50	200	0.2	34.11	34.226	15	54.852	13
16th	152	50	400	0.2	136.2	25.772	15	55.618	12

LSTM – Differentiation Only Language model

Position	Numb	Embeddings	Hidden Units	Dropout	Time	PLL Train	Best epoch	PPL Test	Best epoch
1st	47	300	200	0	32.99	21.801	15	54.489	9
2nd	143	300	200	0.2	32.81	29.656	15	55.126	11
3rd	95	300	400	0	93.46	16.472	15	55.299	8
4th	191	300	400	0.2	137	21.659	15	55.337	11
5th	83	200	400	0	136.6	17.18	15	55.47	8
6th	35	200	200	0	58.03	24.18	15	55.579	11
7th	179	200	400	0.2	106.4	24.025	15	55.966	9
8th	131	200	200	0.2	72.23	31.167	15	55.972	11
9th	23	100	200	0	51.68	27.008	15	56.794	10
10th	119	100	200	0.2	31.95	34.603	15	57.708	15
11th	71	100	400	0	135.9	20.576	15	58.149	8
12th	167	100	400	0.2	134.3	26.794	15	58.316	12
13th	11	50	200	0	51.81	31.631	15	60.432	12
14th	59	50	400	0	135.3	23.077	15	60.675	9
15th	155	50	400	0.2	138.2	29.66	15	61.759	10
16th	107	50	200	0.2	34.06	39.154	15	61.79	12