Word Prediction Using Recurrent Neural Networks

Kiran Nanjundaswamy
Northeastern University, Boston, MA, 02115
nanjundaswamy.k@husky.neu.edu

Abstract

Word prediction, or *language modeling*, is the task of predicting the most likely words following the preceding text. It can be used to suggest the next word entered in a text, text generation and aid in ambiguity in speech and handwriting recognition.

Word prediction has been used to provide autocomplete sentences in google search since a long time. Today many of the search platforms have word prediction or search autocomplete in built. Word prediction has also been used in messaging apps to help generate text quicker. When a user inputs texts, the app will predict the next word in the sentence based on the previous words. It is known that neural networks produce more accuracy when compared to the n-gram model. In this project, we will be looking at designing a model for word prediction using Recurrent Neural Networks. But this is not the goal of the project. The goal of this project is to determine the accuracy of the network model when we introduce plain vanilla text from books collected from Project Gutenberg as the training dataset. Next, we will observe the changes in accuracy when the data from blogs is added to the training dataset. Once the model was trained we achieve good distinctive results. The model produced a maximum accuracy of 15.49% when introduced to mixed data sets and an accuracy of 19.6% when only plain text from book was used for training. The procedure and details are further discussed.

Introduction

Previously n-gram models were used in the word prediction applications. This model predicts the next word based on a limited number of previous words. The models store trigrams and bigrams and uses these to predict successive words. The advantage of using this model is that we do not need to train the model. Preprocessing is required to generate the n-grams which does not take much time. The disadvantage is that the model has an average 30% accuracy. Additionally, it fails to predict words which have a reference to the context of the sentence like "I live in France. I speak *French*" as the context is lost.

The generation of a likely word given prior words goes back to Claude Shannon's work on information theory. It was based in part on Markov models introduced by Andrei Markov where counts of encountered word sequences are used to estimate the conditional probability of seeing a word given the prior words. These so-called n-grams formed the basis of commercial word prediction software in the 1980's, eventually supplemented with similar syntax and part of speech predictions.

More recently, word embedding's were used in the word prediction models. Word embedding's have words represented by a vector in some low-dimensional vector space. These can better handle data sparsity and allow more of the context to affect the prediction, as they can represent similar words as being close together in vector space. Since 'dog' and 'cat' are like each other, the behaviors of a dog like sleeping, running, chasing etc. can be mapped to a cat when predicting words of a sentence involving animals.

As mentioned previously we will be using a Recurrent Neural Network to build a model which can predict the next word in a sentence. The neural network learns word vectors as it is trained. The pre-trained word embedding's GloVe (Pennington 2014) can be used in place of the input layer to save on training time. The rest of the neural network learns a function that maps a sequence of input word vectors to the probability of the next word in the sequence, over the whole vocabulary. This is a problem of supervised learning, as any text can be used to train and evaluate the models. In the following sections, we will try to understand the working of neural nets and how to build a network model and discuss the training and testing phases.

Recurrent Neural Networks

Traditional neural networks do not have persistence. They do not keep memory of the past events to determine the current event. If we want to classify what kind of event is happening at every point in a movie. The network cannot infer from previous events in the film to inform later ones. For this we need a recurrent neural network. These networks have loops in them allowing information to persist

over time. Recurrent Neural Networks (RNNs) are very powerful sequence models but do not enjoy widespread use because it is extremely difficult to train them properly. In specify we will be using a grated mechanism is the neural network which outperforms LSTM's and which is easier to train. The details of GRU's and LSTM's will be discussed later.

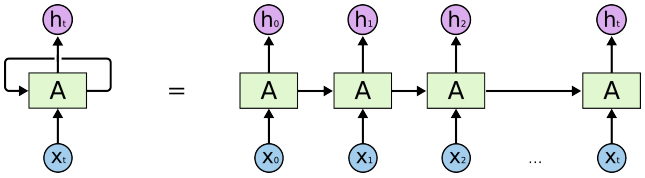

Figure 1: A graphical representation of recurrent neural network shown as one unrolled network.

In the figure, we can see a chuck of recurrent neural network which is unrolled to reveal a chain like structure. We have an input 'X' as some point in time say t. This when given to the network will produce an output 'H' at time t. A loop allows information to be passed from one step of the network to the next. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The chain like structure is seen to have data at every point instance of time t. This shows us that recurrent neural networks are intimately related to sequences and lists

Keras and Tensorflow library

To help in designing of these neural networks, many libraries have been developed to make the task simpler. Some of the libraries are Theano, Caffe, Chainer, DeepNet, DeepPy, ConvNet, Torch, MxNet, Lasagne. Google's TensorFlow library is one of the popular and open source library for numerical computation using dataflow graphs. A primary benefit of TensorFlow is distributed computing, particularly among multiple GPU's. Nodes in the graph represents mathematical operations while the graph edges represent the multidimensional data arrays communicated between them. Google has also designed processing units called Tensor Processing Units (TPU's) which are aligned to work perfectly with TensorFlow library.

Keras is an open source neural network library written in Python. It can run on top of Tensorflow or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being minimal, modular and extensible. It was developed with a focus on enabling fast experimentation. It runs seamlessly on CPU and GPU. We use Keras as we are focusing on doing a quantitative analysis of the changes in accuracy with dataset and not on the development of a neural network.

Datasets

For this project, we will be using plain vanilla text from books digitized by the Gutenberg Project (Gutenberg 2016). These are a collection of novels from famous authors over time. The data can be found in text format and can be easily cleaned and preprocessed from our purpose. We will be using 20 books from this data set which contains counts of words ranging from 10,000 to 150,000.

We also be using blogs data set to do a comparative analysis of the data. The blogs data was retrieved from Coursera-SwiftKey data set which contains 3 unique datasets of blogs, new and twitter data each of size 200MB. We do not need such huge amount of data. We will be using only 10% of the blogs data set.

For this project, we will also be using 100-dimensional word vectors from GloVe (Pennington 2014) which contains around 400,000 words. GloVe is an abbreviation for Global Vectors for word representation. As mentioned earlier these can be used in place of the input layer to save on training time. They have only become available recently due to the computational complexity of calculating them for large vocabularies

"Indeed, I should have thought a little more. Just a trifle more, I fancy, Watson. And in practice again, I observe. You did not tell me that you intended to go into harness."

"Then, how do you know?"

Extract from Adventures of Sherlock Homes plain vanilla text of Project Guttenberg

A neighbor recommended "Chasing Fireflies" by Charles Martin. I'd never read anything by Martin before, although he was this neighbor's favorite author. "Fireflies" is the story of Chase, a journalist who never knew his parents and was adopted by "Unc," a colorful resident of a small Georgia town whose sayings and kind heart make him a memorable character. Chase is investigating the story of another orphan, this one a mute boy pushed out of a car on train tracks just before a train hit and killed his loser custodial "parent."

Extract from blogs data of Coursera-SwiftKey data set

the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459 0.28787 -0.06731 0.30906 -0.26384 -0.13231 -0.20757 0.33395 -0.33848 -0.31743 -0.48336 0.1464 -0.37304 0.34577 0.052041 0.44946 -0.46971 0.02628 -0.54155 -0.15518 -0.14107 -0.039722 0.28277 0.14393 0.23464 -0.31021 0.086173 0.20397 0.52624 0.17164 -0.082378 -0.71787 -0.41531 0.20335 -0.12763 0.41367 0.55187 0.57908 -0.33477 -0.36559 -0.54857 -0.062892 0.26584

Extract from GloVe dataset for the word vector representation of the word 'the'

Background

Artificial Neural Networks

Artificial Neural Networks was a concept developed in 1943. It was modeled after the structure of the human brain. The idea saw a lot of research in the early days but declined due to the technical limitations of the systems which existed before 1990's. The computational power of the system was not very high and it would take a very long time to train a simple neural network. A breakthrough in training time was achieved when the backpropagation algorithm was discovered. Here the data would be input to the network and the error rate would be calculated at the output layer. Then the weights would be changed to minimize the error and provide better outputs. Attempting to minimize the error led to neural nets which were more efficient and quicker to train. With the advancement of technology and development of many computationally powerful systems, neural nets saw a rise again. Neural nets are mostly used for deep learning which is an area of machine learning in modelling relationships which are non-linear and complicated.

A simple neural network contains nodes through which data is inputted to the system. The data is propagated to a set of hidden layers which is a layer consisting of multiple nodes. A network can have one or many hidden layers. The nodes in each layer are connected to nodes in the incoming and outgoing layers by different, adjustable weights. The data is passed through the hidden layers and eventually reaches the output layer where the data can be interpreted as different results. The network shown below consists of 3 input layer and only 1 hidden layer consisting of 4 nodes and 2 nodes in the output layer. The nodes in each layer are connected to the nodes in the other layer by weights. A bias is also added to each node in the network.

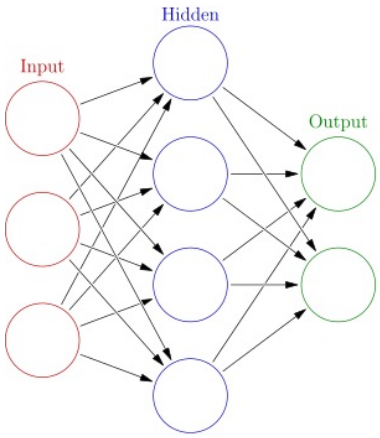

Figure 2: Simple artificial neural network with 3 input nodes, 4 hidden nodes and 2 output nodes.

Backpropagation

Backpropagation is a method of training artificial neural networks. The algorithm repeats a two-phase cycle, propagation and weight update. Before training the weights are set randomly.

• In the first phase, the input is propagated forward through the network layer by layer until it reaches the output layer. The output of each neuron in the network is given as the weighted sum of all its inputs.

• The network output is compared to the desired output and the loss function and error values are calculated. A common way to measure error is given as

• The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output. the loss function over n training examples can be written

• These error values are used to calculate the gradient of the loss function with respect to the weights in the network. The gradient calculated as a partial derivative of the output

• In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, to minimize the loss function.

On completion of training using backpropagation, the neurons in the intermediate layers organize themselves in a way that different neurons recognize different characteristics of the input space.

Backpropagation Through Time

Backpropagation through time (BPTT) is a gradient-based technique for training certain types of recurrent neural networks. The training data for BPTT should be an ordered sequence of input-output pairs. BPTT begins by unfolding a recurrent neural network through time. The training is like a feed-forward neural network with backpropagation discussed earlier, except that the training patters are visited in sequential order. This technique tends to be significantly faster for training recurrent neural networks than general-purpose optimization techniques. In recurrent neural networks, the local optima is a significant problem. The feedback in such networks tends to create chaotic responses in the error surface which cause local optima to occur frequently.

LSTMs & GRUs

Long short-term memory is a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. They remember information for long periods of time. LSTMs also have chain like struc-

tures, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. We can use LSTM's in sequential data such as word prediction as we need to keep track of the previous words in the sentence. The details of this model are beyond the scope of this project.

In this project, we will be using Gated recurrent units. GRUs are a gating mechanism in recurrent neural networks, introduced in 2014. GRUs have fewer parameters and thus may train a bit faster or need less data to generalize. On the other hand, if you have enough data, the greater expressive power of LSTMs may lead to better results.

Supervised and Unsupervised Learning

Machine learning algorithms fall into two categories, supervised or unsupervised learning. Problems such as classification and regression use supervised learning and problems related to clustering and association use unsupervised learning. Here we will be using supervised learning.

Supervised learning is a concept where you have an input variable X and an output variable Y and we use an algorithm to learn the mapping function between X and Y such that Y = f (X). The idea is to reduce the error in the mapping function such that when the algorithm is fed a previously unseen data, it can predict the correct output variable Y for the data. In supervised learning, we have a set of training data and labels associated with the data which categorize them correctly. The algorithm learns to categorize the input by making predictions and understanding the error based on the associated labels. The learning stops when the algorithm achieves an acceptable level of performance.

In unsupervised learning, we are only given the input X and no corresponding output variables. The goal is to model the underlying structure or distribution of the data to learn more about the data given. Here there is no correct answer and we do not know the error in the result achieved. The algorithms are left to discover and present the interesting structure of data

Project description

Data Preprocessing

Before we train the network, we will need to perform some data processing. The raw data contains Unicode characters which need to be decoded as python string functions will not work on Unicode and characters like "," etc. cannot be processed by NLTK library for natural language processing. We also do not require any punctuation as we are not checking for the grammatical syntax of the sentences. We are only interested in the correct context of the text generated. NLTK is a natural language tool kit built for python. It contains easy to use interfaces for corpora along with a suite of text processing libraries for classification, tokenization, stemming, tagging etc. We also convert all the text to lower case for ease.

We first read the data from the file, convert it to lower case and then replace all "," with asci characters and replace the punctuation. Once this first phase of processing is done we get a string with all the processed text of the file. Now we will break the text in to a list of sentences and we will shuffle them up to achieve some random order which is better for training of a neural network. Next, using NLTK's tokenization functions we will tokenize the data and find the N most common words in the text. This will return us a list of tuples containing the word and the frequency of the word

Next, we take only the words in this list and we build an index-word dictionary mapping and a word-index dictionary mapping. We require the word to index mapping as we will be building a sequence array which will be used for training the neural network. We require the index-word mapping as we want to determine given a number, which word should be substituted.

Finally, we require the embedding matrix which is retrieved from the GloVe dataset. The data in the dataset is in the form of word followed by array of the word vector coefficients. Each line is read and broken into parts and stored in a dictionary to be used later to build the embedded layer for the neural network.

When we are checking for the accuracy of prediction using only the plain vanilla text of books we ignore the blog text data. If we are checking the accuracy including both then we will process both the files, combine the text from both the files and continue the process as mentioned above. Next, we move on to building and training the network.

Building the Recurrent Neural Network

We will use the Keras library with a TensorFlow backend to help in building the neural network. The basic architecture of the RNN is an embedding input layer, one or more hidden layers, a densely-connected output layer, and a softmax layer to convert the outputs to probabilities. The neural network learns word vectors as it is trained.

The GloVe embedding matrix is used in place of the input layer to save on training time. The embedding matrix is initially set to an array of 0s of size equal to the number of vocab we are interested in and 100 dimensions wide. Next, we get the index of each word from the dictionary we previously built and will substitute the word vector value for the same word in the embedding matrix at the location equal to the words index value. Each word in the embedding matrix will correspond with a row in the table

and the distance between any two words gives their similarity. Eventually the neural network effectively learns a function that maps a sequence of input word vectors to the probability of the next word in the sequence, over the whole vocabulary. Once we have the embedding matrix this is used as the weights of the input embedding layer.

Next, we design the network to have 3 hidden layers show as *gru* in the figure below. The number of nodes in each of these hidden layers is set to 100. Each of the hidden layer is configured to be a grated recurring unit as previously discussed. We also add dropout to these layers equal to 0.1. To improve the accuracy of the model we apply dropout during training. During this process, random sets of input units are set to 0 at each update, which helps to prevent overfitting. This also reduces the number of parameters required to be learned in a fully connected layer. It increases the robustness of the model by forcing it to learn multiple ways to represent the same patterns. We will see another mechanism to avoid over fitting later. All the three layers are uniform initialized with weights. The first two hidden layers are set to have return sequences, which means that these later will feed data to the next recurrent layer in the network.

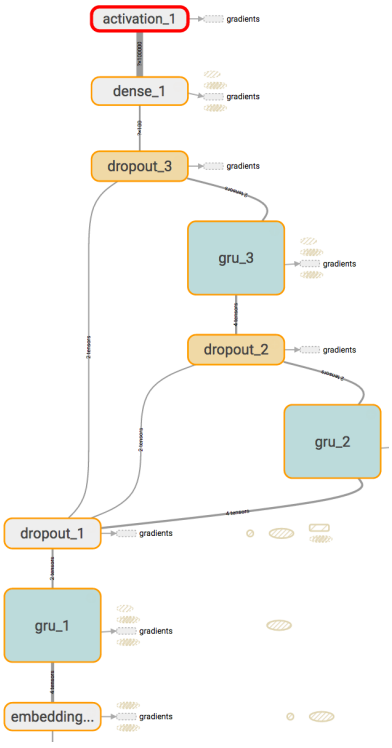

Figure 3: Graphical representation of the network as shown by TensorBoard

The output layer shown as *dense* in the figure is a densely-connected layer of nodes. The number of nodes is equal to the number of vocabularies being used training. In this layer, each word in the vocabulary votes on how likely it thinks it will be next, based on the context (current + previous words). This is calculated with the help of the values which resulted from values inputted into the embedding layer. Next, the softmax output layer shown as *activation* converts these scores into probabilities, so the most likely next word can be found for a given context.

We evaluate the performance of the model by checking the accuracy. The average number of predictions where the highest probability next word matches the actual next word is determined to be the accuracy. To learn the parameters for training the network uses a loss function called categorical cross-entropy loss. Backpropagation through time is used to update the weights based on the amount of error between the output and the true values.

Neural networks can over fit to the training data if they are trained for too many epochs. Another precaution to prevent over fitting on top of dropout is early stopping. Early stopping triggers are set on the model which will stop training if the validation accuracy does not improve for a certain number of epochs.

Training and Results

Now that the network model has been set up we can train the neural network to predict words in a sentence. First, we need to feed the data to the network. The tokenized data is split into training, validation, and test sets. The test set was set to 10,000 tokens, and the validation set to 10% of the training set, which amounts to around 2700 tokens out of 270,000 tokens.

The training was performed for 50 epochs for both sets of data, one containing only the plain vanilla text and the other containing both the plain vanilla text and the blogs data set. Some very interesting results were discovered which are discussed below.

Now we must understand that the data sets we are using are very different. The books contain English in the written format which was used while writing booking in the 19th – 20th century. The blogs data set contains English in the format as spoken in the 21st century. Now mixing these two should clearly have some distinct results.

Each epoch takes roughly around 30-45 mins to complete. When the mixed dataset containing both the plain text and the text from blogs data was sent to the network, the network stopped training after 18 epochs. After this point, the network was not able to achieve a better accuracy and because of our network parameter early stopping, training was stopped. In the 1st epoch the network generated text such as the one seen below.

Epoch 1

"the same of a man of a little and i was the little and a few and i have not".

It did predict some of the words corrected but still wasn't doing good enough. On further training, the network could generate the following sentences

Epoch 10
"the same year is not to the point of this agreement to get a little little little little bit in"

Epoch 18
"each format could not allow him a new work and i am a little of a good way to find"

Final Epoch
"are forced for the first week ago the count was a few weeks ago and i have been a few"
"neck was forced to keep a knife he began the of his hands with his face the same time i"
"hand that the second was waiting to be a few weeks to be in the room and then i had"
"came into the table floor with a candle from my own and he was so well then he was the"
"and have done a bit and i am afraid of my current person i have been a little of a"
"to include the medium and raw and the full man had a good and the most in his wife"

As it can be seen with every epoch a little bit of improvement was seen in the generated sentences. Repetitive training was done by saving the model and starting off from previous saved point once the network stopped training to see the best possible accuracy. The maximum accuracy that the network could achieve with this dataset was 15.49%. The below graph shows the change in accuracy with respect to the number of epochs

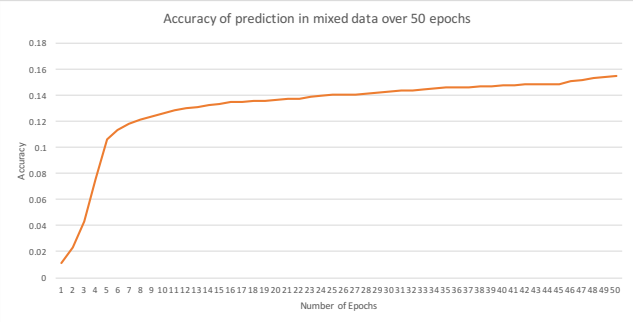

Figure 4: Graph of the accuracy of prediction for mixed data set

With a decrease in loss, the network tends to perform better. The best minimum loss in the network which was achieved by repetitive training was 5.12%. The below graph shows the change in loss with respect to the number of epochs

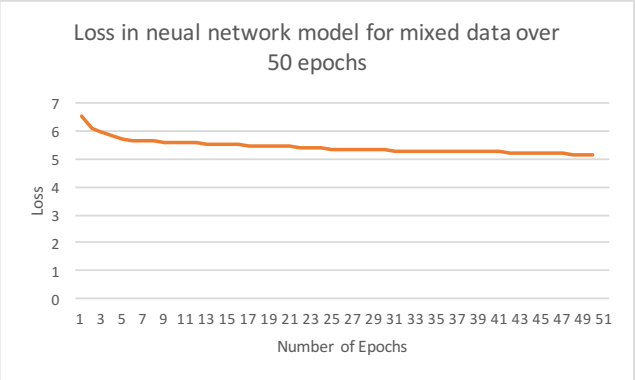

Figure 5: Graph of the loss of in network for mixed data set

When the plain text was sent to the network without mixing it with the blog data, the network stopped training after 22 epochs. But the network performed much better than the mixed data set.

Epoch 1
"the and a room of the and the and the room and the room of a room in the room"

Epoch 10
"corridor dusk below with the corner but she had a few of the world and i had not a time"

Epoch 18
"the subject is the bottle of finding but i am glad i could have a good of the of the"

Final Epoch
"neck was forced to keep a knife he began the of his hands with his face the same time i"
"hand that the second was waiting to be a few weeks to be in the room and then i had"
"came into the table floor with a candle from my own and he was so well then he was the"
"and have done a bit and i am afraid of my current person i have been a little of a"

As seen in the above sentences we can clearly say that when the network was fed non-mixed data, it performed better than with mixed data. With repetitive training the best achievable accuracy was 19.6% and lowest recorded loss was 4.31% The graph shows the change in accuracy and loss over the number of epochs.

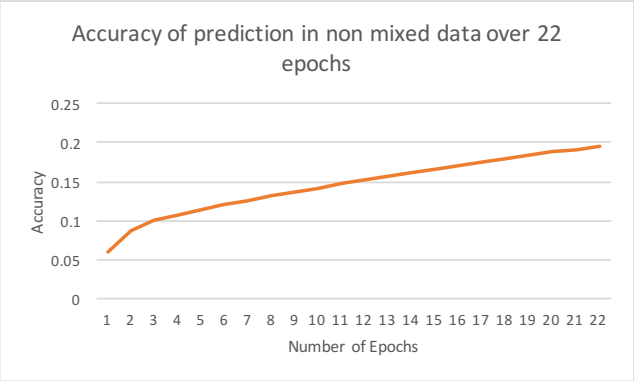

Figure 6: Graph of the accuracy of prediction for non-mixed data set

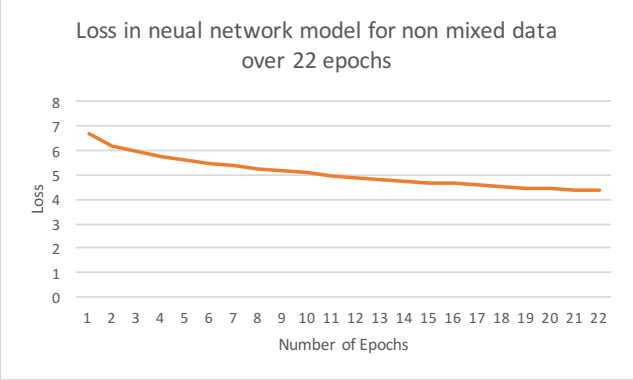

Figure 6: Graph of the loss of in network for non-mixed data set

Below is a comparative study of the changes in loss and accuracy from for mixed as well as non-mixed data sets.

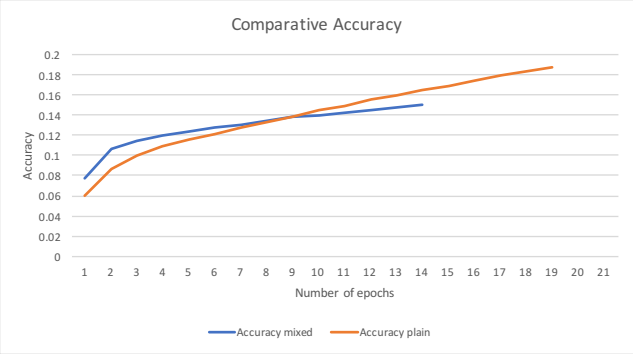

Figure 7: Comparison of accuracy in network between mixed and non-mixed (plain) data sets

Figure 8: Comparison of loss in network between mixed and non-mixed (plain) data sets

From the above observations, it can be clearly determined that the mixing of the data caused the data accuracy to change in the network prediction.

TensorBoard allowed from a more detailed study on word vectors. Next, we will discuss about this and gain a little more understanding of the data inputted to the network.

Word Vectors

As previously discussed, we used word embedding from the GloVe data set to reduce our training time. With the help of TensorBoard we can visualize this to better understand how the words were represented and how relationship between them were maintained.

In the below figure, we can visualize the embedding matrix which was collected from tensorboard. The words which were loaded from the GloVe data set were matched with the words which were tokenized after preprocessing and the resultant was this embedding matrix.

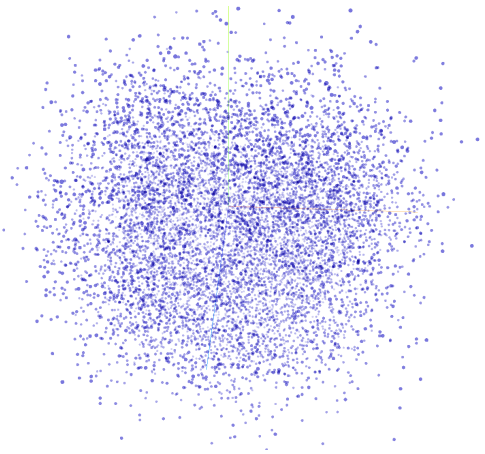

Figure 9: Graphical representation of the embedding matrix as a PCA as shown by TensorBoard

In figure 10, we can see a closer view of these points in space and we can gain a little more understanding. The highlighted number 27 is equivalent to 'we', 10 is equivalent to 'was' and 19 is equivalent to 'his' and these are seen close to each other in the figure. These words are closely correlated to each other and this can be clearly seen. The other numbers 255, 347 and 268 are equivalent to 'seen', 'child', 'brought'. Sentences can contain words such as 'he brought', 'we brought', 'was brought', 'was seen', 'his child'. So, we can clearly understand why these points are seen close to the previously seen points. There exists a close relationship between these words and this is what is represented in the embedding matrix which helps us to predict the next words in a sentence.

In figure 11, we can see another perspective of this relationship, but after it has been passed through the network and the dimensions have been reduced.

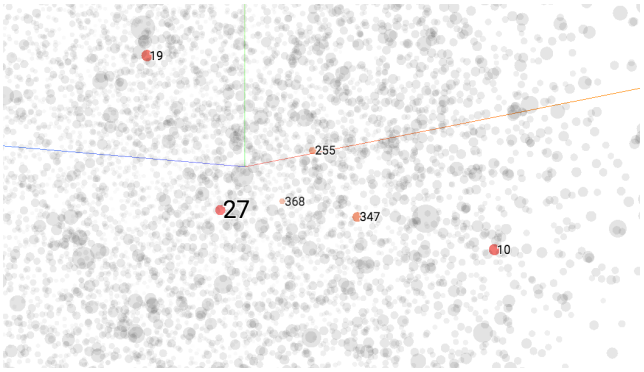

Figure 10: Word vectors which are closely related are closer in the embedding matrix as shown by TensorBoard

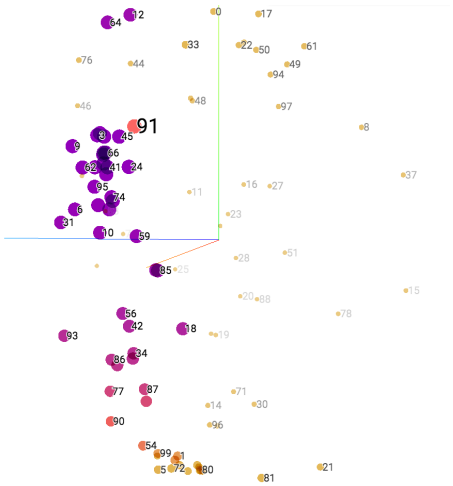

Figure 11: Reduced dimension of words vectors after they were passed through the network as shown by TensorBoard

Conclusion

Recurrent Neural Networks work well when used to predict words in a sentence. RNN's can retain more history and using word vectors instead of discrete tokens allows us to generalize and make predictions of words with similar characteristics. These properties show us that neural networks can perform better than n-gram models in predication, but this is achieved at the cost of a longer training time. Another advantage of using a neural net is that once trained, it is small enough to fit on a mobile or a tablet and can be used for word prediction. The SwiftKey app has been using neural nets to predict the next word while typing since quite some time.

Now considering our project, we can say that the data set chosen to train the neural network also plays a very important role in the accuracy of the model. We used a data set of plain English text from novels from the 19th – 20th century as a baseline. With this we could achieve a 19.6% accuracy in prediction. This is quite high for a network which was trained for less than 11 hours. But once the data was mixed with the blogs data set, which contained spoken English text, the accuracy dropped to 15.49%. That is a 4.11% drop in accuracy. This is presumably due to the variations in the way the English text was spoken and represented in different centuries.

Now it may seem that our model has a very low accuracy and would not perform well in the outside world. The aim of this project was to do a comparative analysis on the datasets being used for training the model and how it would affect the accuracy of the model. We have not explored the best parameters and best architecture for the neural network. Firstly, we can train the model with more data. But this would again increase the time required for training. We should try to train the data on a GPU to help in faster processing. We can use methods such as the hierarchical softmax or differentiated softmax to help in reducing the computation time. And, we can work with different architectures of the network model to achieve better results. Designing a neural network is a science of its own.

References

1. TensorFlow. (2017). TensorFlow. [online] Available at: https://www.tensorflow.org/ [Accessed 21 Apr. 2017].
2. Keras.io. (2017). Keras Documentation. [online] Available at: https://keras.io/ [Accessed 25 Apr. 2017].
3. Nltk.org. (2017). Natural Language Toolkit — NLTK 3.0 documentation. [online] Available at: http://www.nltk.org/ [Accessed 25 Apr. 2017].
4. Colah.github.io. (2017). Understanding LSTM Networks -- colah's blog. [online] Available at: http://bit.ly/2oFtfZt [Accessed 25 Apr. 2017].

5. En.wikipedia.org. (2017). Gated recurrent unit. [online] Available at: http://bit.ly/2oFpR0w [Accessed 25 Apr. 2017].
6. En.wikipedia.org. (2017). Recurrent neural network. [online] Available at: http://bit.ly/2oFsv6t [Accessed 25 Apr. 2017].
7. Brownlee, J. (2016). How to Check-Point Deep Learning Models in Keras - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oFssrp [Accessed 25 Apr. 2017].
8. Brownlee, J. (2016). Save and Load Your Keras Deep Learning Models - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oFqzuS [Accessed 25 Apr. 2017].
9. Brownlee, J. (2016). Text Generation With LSTM Recurrent Neural Networks in Python with Keras - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oFsQ9f [Accessed 25 Apr. 2017].
10. Keunwoo Choi. (2016). Keras callbacks guide and code. [online] Available at: http://bit.ly/2oFuWFY [Accessed 25 Apr. 2017].
11. Rpubs.com. (2016). RPubs - Next Word Prediction (Natural Language Processing Project). [online] Available at: http://bit.ly/2oFhRwI25 Apr. 2017].
12. Frachon, L. (2017). Predictive Text Application - Milestone Report. [online] Pubs.thetazero.com. Available at: http://bit.ly/2oFBzbp [Accessed 25 Apr. 2017].
13. GitHub. (2017). sjuvekar/neural-network-coursera. [online] Available at: http://bit.ly/2oFwwrL [Accessed 25 Apr. 2017].
14. GitHub. (2017). karpathy/char-rnn. [online] Available at: http://bit.ly/2oFrVWk [Accessed 25 Apr. 2017].
15. Brownlee, J. (2016). Text Generation With LSTM Recurrent Neural Networks in Python with Keras - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: http://bit.ly/2oFsQ9f [Accessed 25 Apr. 2017].
16. GitHub. (2017). Kyubyong/word_prediction. [online] Available at: http://bit.ly/2oFrVWk [Accessed 25 Apr. 2017].
17. WildML. (2015). Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs. [online] Available at: http://bit.ly/2oFmcjn/ [Accessed 25 Apr. 2017].
18. Gutenberg.org. (2017). Top 100 - Project Gutenberg. [online] Available at: http://bit.ly/2oFynN6 [Accessed 25 Apr. 2017].
19. Lars Eidnes' blog. (2015). Auto-Generating Clickbait With Recurrent Neural Networks. [online] Available at: http://bit.ly/2oFp13R/ [Accessed 25 Apr. 2017].
20. Rosenthal, J. (2017). Tutorial: Recurrent neural networks - Nervana. [online] Nervana. Available at: http://bit.ly/2oFtAuZ [Accessed 25 Apr. 2017].
21. T. Mikolov, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur, "Extensions of recurrent neural network language model," 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Prague, 2011, pp. 5528-5531.
22. Citeseerx.ist.psu.edu. (2017). [online] Available at: http://bit.ly/2p0tgdC [Accessed 25 Apr. 2017].
23. Machinelearning.wustl.edu. (2017). [online] Available at: http://bit.ly/2p0qi90 [Accessed 25 Apr. 2017].