RECURRENT NEURAL NETWORK BASED LANGUAGE MODELS

A Project Report

submitted by

SRIKANTH PRABALA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **RECURRENT NEURAL NETWORK BASED LANGUAGE MODELS**, submitted by **Srikanth Prabala**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. S Umesh
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th May 2016

ACKNOWLEDGEMENTS

I would like to thank Prof Umesh for giving me an opportunity to work with him and for assisting me with my project. I would also like to thank my lab-mates especially Sandeep Kothinti for assisting me with Kaldi tool kit. I would like to thank Tomas Mikolov for his RNNLM toolkit on which the experiments are done. Finally I would like to thank my parents and friends for their support during this period.

ABSTRACT

KEYWORDS: Recurrent Neural Network; Language model; N-gram.

Statistical language models are crucial part of automatic speech recognition, machine translation etc. Traditionally N-gram based models were used for language modeling. With the advancement in computational capacity (Moore's law) neural networks have captured the machine learning market completely. Despite huge advances in machine learning research N-gram based models have been the state of the art in language modeling. Recent application of neural networks to language modeling have shown good improvements compared to the N-gram models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

RNN	Recurrent neural network
LSTM	Long short-term memory
WER	Word error rate
WSJ	Wall street journal
NLP	Natural language processing
BPTT	Back propagation through time

CHAPTER 1

INTRODUCTION

1.1 Overview

A statistical language model is a probability distribution over sequence of words. Given a sequence of length m, a language model assigns the probability $P(w_1..w_m)$ to the whole sequence. Language modeling has various applications in various fields like speech recognition, handwriting recognition, machine translation etc.

In speech recognition (matching sounds with words) the language model provides context to distinguish between sounds that sound similar but have different meaning. For e.g. 'recognize speech' and 'wreck a nice beach' are pronounced in almost similar way and they beat the acoustic model during recognition. This can be corrected using language model.

Artificial neural networks are family of models inspired by biological neural networks in the brain. These are used to approximate random functions with large number of inputs. These are known to find complex pattern in the data with sufficient training and can also be used to map data from higher dimensions to lower dimensions.

1.2 Motivation

Language modeling has become an interdisciplinary field when it comes to natural language processing with many applications in most of NLP's sub-fields.

1.3 Organisation

This thesis involves the application of recurrent neural networks a type of artificial neural network for language modeling. First we will learn about two types of language models: baseline N-gram and RNN based language model.

Then we would evaluate the performance of RNN based language model and then compare it with the tri-gram and four-gram models. Finally we would combine both the models to get the maximum accuracy.

CHAPTER 2

LANGUAGE MODELS

2.1 N-gram models

In this model the basic assumption we make is that the probability of a word only depends on the previous n-words. Depending on the value n the model can be called as uni-gram, bi-gram, trigram etc.

The probability of word sequences is calculated using chain rule:

$$P(w) = \Sigma_{i=1}^{N} P(w_i|w_1..w_{i-1})$$

N-grams are the most frequently used language models, these are basically word co-occurrence frequencies. The maximum likelihood estimate of probability of word W in context K is computed as

$$P(W|K) = \frac{C(KW)}{C(K)}$$

where C(KW) is the number of times that the KW word sequence has occurred in the training data.The value 'N' in the n-gram model depends on the number of words we consider in the context K. |K| = 2 is trigram, |K| = 3 is four-gram etc.

Smoothing is done to reduce some erroneous over-estimations specific to the training data. For example,if the sentence

```
Party is on Musk
```

occurs in the training data out of a million words, it's frequency is still overestimated. To reduce this effect smoothing is done on the data. In this case Katz smoothing is done.

$$P_{bo}(w_i/w_{i-n+1}...w_{i-1}) = \begin{cases} d_{w_{i-n+1}..w_i} \frac{C(w_{i-n+1}..w_i)}{C(w_{i-n+1}..w_{i-1})}, & \text{if } C(w_{i-n+1}..w_i) > k \\ \alpha_{w_{i-n+1}..w_{i-1}} P_{bo}(w_i/w_{i-n+2}...w_{i-1}), & \text{otherwise} \end{cases}$$

$C(x)$ is the count or frequency of x and w_i is the i^{th} word in the context.

Essentially, this means that if the n-gram has been seen more than k times in training, the conditional probability of a word given its history is proportional to the maximum likelihood estimate of that n-gram. Otherwise, the conditional probability is equal to the back-off conditional probability of the "(n-1)-gram".

To compute α, it is useful to first define a quantity β, which is the left-over probability mass for the (n-1)-gram:

$$\beta_{w_{i-n+1}..w_{i-1}} = 1 - \sum_{w_i:C(w_{i-n+1}..w_i)>k} d_{w_{i-n+1}..w_i} \frac{C(w_{i-n+1}..w_i)}{C(w_{i-n+1}..w_{i-1})}$$

Then back-off weight α, is calculated as:

$$\alpha_{w_{i-n+1}..w_{i-1}} = \frac{\beta_{w_{i-n+1}..w_{i-1}}}{\sum_{w_i:C(w_{i-n+1}..w_i)\leq k} P_{bo}(w_i/w_{i-n+2}...w_{i-1})}$$

The above formula applies if there is a valid n-1 gram else it uses Katz estimate for n-2 gram and so-on.

d is discounting found from Good-Turing estimation.

The idea of Good-Turing estimation is reallocate probability mass of n-grams that occur r+1 times in the training data the n-grams occurring r times, in particular to the n-grams that are not seen.

For each count r we compute an adjusted count r^*:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

where n_r is the number of n-grams seen r times. Then:

$$P_{GT}(x : c(x) = r) = \frac{r^*}{N}$$

where $N = \sum_{r=0}^{\infty} r^* n_r = \sum_{r=1}^{\infty} r n_r$

Despite the huge success of N-gram models, with basic analysis of the working of N-gram models one could see that they fail to take the context of the words prior to the 'N' words. For example

```
THE WATER IN THE OCEAN IS BLUE
```

```
THE WATER IN THE OCEAN TO THE EAST OF CHENNAI IS BLUE
```

The two sentences above almost convey the same meaning. In both the sentences the color BLUE is implied by OCEAN, but for a tri-gram model the context of ocean is captured only in the first case. In the second case the words TO THE EAST OF CHENNAI hinder the context to pass through when the model reaches the word BLUE. This is the primary drawback of N-gram models.

2.2 Recurrent neural networks

A recurrent neural network is a class of artificial neural network with a directed cycle between the connections. It creates an internal state inside the network.

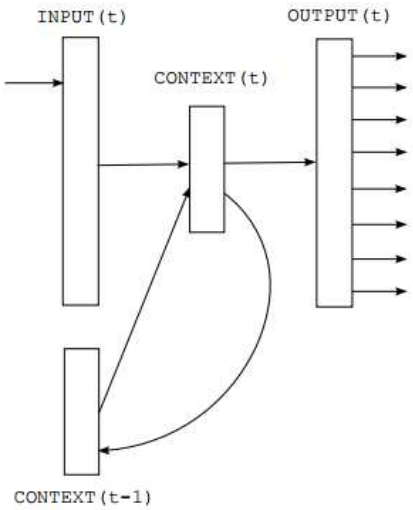

Figure 2.1: Simple recurrent neural network

The memory of the internal state stores the context based information. This helps the network to estimate the word probabilities in a better way.

The network has an input layer x, hidden layer s (also called context layer or state) and output layer y. Input to the network in time t is $x(t)$, output is denoted as $y(t)$, and $s(t)$ is state of the network (hidden layer). Input vector $x(t)$ is formed by concatenating vector w representing current word, and output from neurons in context layer s at time $t-1$. Input, hidden and output layers are then computed as follows:

$$x(t) = w(t) + s(t-1)$$

$$s_j(t) = f(\sum_i x_i(t) u_{ji})$$

$$y_k(t) = g(\sum_j s_j(t) v_{kj})$$

where $f(z)$ is a sigmoid activation function

$$f(z) = \frac{1}{1+e^{-z}}$$

and g(z) is softmax function

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

For initialization, $s(0)$ can be set to vector of small values, like 0.1 - when processing a large amount of data, initialization is not crucial. In the next time steps, $s(t+1)$ is a copy of $s(t)$. Input vector $x(t)$ represents word in time t encoded using 1-of-N coding and previous context layer - size of vector x is equal to size of vocabulary V plus size of context layer.

The RNN is trained using the standard back propagation algorithm or back propagation through time algorithm(BPTT) with the objective function being minimising the cross entropy function.

```
Pseudo Code for BPTT:
Back_Propagation_Through_Time(x, y):
    Unfold the network to contain k instances of f
    do until stopping criteria is met:
        s = the zero-magnitude vector;
        for t from 0 to n - 1
            Set the network inputs to
            s, x[t], x[t+1], ..., x[t+k-1];
            p = forward-propagate the inputs over the
            whole unfolded network;
            calculate error between y[t+k] and p;
            Back-propagate the error, e, back across
            the whole unfolded network;
            Update all the weights in the network;
            Average the weights in each instance of f
            together,so that each f is identical;
            s = f(s);
```

To explain in simple terms BPTT is extending back-propagation to an RNN unfolding it k time stamps and treating it as a feed forward neural network and finally averaging over the k weights.

CHAPTER 3

EXPERIMENTS

The experiments in this section are done on Wall street journal data base and some additional data from Google's 1 billion words database. The additional words(sentences) taken from Google's database contain the same vocabulary as in the Wall street journal database.

The vocabulary for this experiment is approximately 18000 words. The results are reported after re-scoring 10 best paths. We have also used 50 nodes in the hidden layer during training.

3.1 Evaluation

Evaluation in the experiments is done on the basic of word error rate. The word error rate is defined as:

$$WER = \frac{min(S + D + I)}{N} * 100$$

It is the minimum number of S+D+I combined where S stands for substitutions, D stands for deletions and I stands for insertions.

3.2 Results

For the columns in the following tables 'α-RNN', α is the weight-age given to the RNN model and $1 - \alpha$ is the weight-age given to the N-gram model while calculating the probability i.e. while estimating the probability of a word the probability of the word is calculated as the weighted mean of RNN model and N-gram model. THe size column indicates size of the training data.

Table 3.1: RNN+Trigram on Dev93

Dev93					
Size	Trigram	0.25-RNN	0.5-RNN	0.75-RNN	RNN
600K	20.67	19.84	19.84	19.93	20.05
1.5M	17.63	16.73	16.58	16.74	16.91
5.3M	17.02	16.01	15.88	16.01	16.08
10M	16.92	15.93	15.86	15.93	16.07

Table 3.2: RNN+Trigram on Eval92

Eval92					
Size	Trigram	0.25-RNN	0.5-RNN	0.75-RNN	RNN
600K	9.6	8.84	8.8	8.97	9.3
1.5M	7.79	6.99	6.76	6.97	7.19
5.3M	7.42	6.58	6.31	6.5	6.71
10M	7.29	6.41	6.31	6.46	6.71

Table 3.3: RNN+Fourgram on Dev93

Dev93					
Size	Fourgram	0.25-RNN	0.5-RNN	0.75-RNN	RNN
600K	20.55	19.68	19.6	19.86	19.96
1.5M	17.68	16.77	16.54	16.72	16.95
5.3M	16.92	16.2	15.9	16.05	16.14
10M	16.84	15.8	15.7	15.82	15.95

Table 3.4: RNN+Fourgram on Eval92

Eval92					
Size	Fourgram	0.25-RNN	0.5-RNN	0.75-RNN	RNN
600K	8.84	8.44	8.57	8.97	9.27
1.5M	7.96	7.14	6.89	6.91	7.08
5.3M	7.51	6.81	6.39	6.48	6.71
10M	7.38	6.8	6.54	6.52	6.84

From the tables we can see that the "pure" RNN model outperforms the base line Trigram and Fourgram models in all the cases.

3.2.1 Size of train data vs WER

Additional data of around 10 million words is added to the existing 600k data obtained from wsj database. The additional data is segregated from google 1 billion word database which contains the existing words. RNN and N-gram language models are being built by adding the additional data stepwise and WER is computed.

Plotting the data from the above tables:

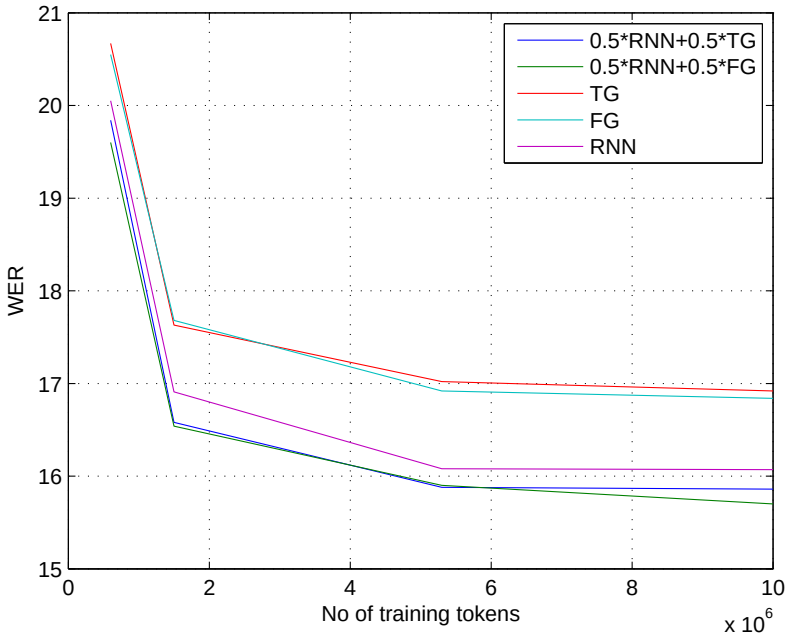

Figure 3.1: No of training tokens vs WER for Dev93

From the above plots we can see that as the number of training tokens increase WER decreases exponentially and after certain number of training tokens error remains constant.

This shows that RNN based language models work best when the amount of train data is huge.

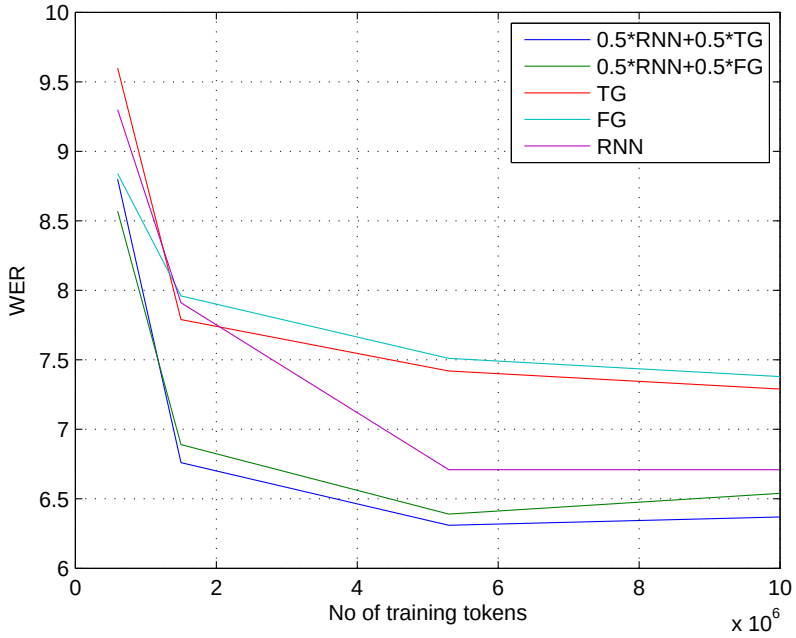

Figure 3.2: No of training tokens vs WER for Eval92

3.2.2 RNN weightage vs WER

In this experiment RNN and N-gram language models are combined with different weight rations and try to find if the accuracy can be improved with a weighted combination of both the models.

Table 3.5: RNN WER for 1.5M words with different α

RNN weight	Dev93 FG	Dev93 TG	Eval92 FG	Eval92 TG
0.1	16.27	16.34	7.02	6.91
0.2	16.08	16.20	6.89	6.56
0.3	16.1	16.13	6.65	6.48
0.4	16.23	16.17	6.61	6.50
0.5	16.26	16.32	6.58	6.48
0.6	16.28	16.42	6.65	6.46
0.7	16.38	16.56	6.69	6.54
0.8	16.55	16.64	6.84	6.61
0.9	16.78	16.74	6.95	6.73
1	16.84	16.84	7.06	6.87

In the above graphs the WER's are fitted with quadratic polynomials to visualize the "performance" of α over the datasets.

For Dev93 the maximum accuracy occurs for $\alpha \approx 0.5$ whereas in eval92 it occours

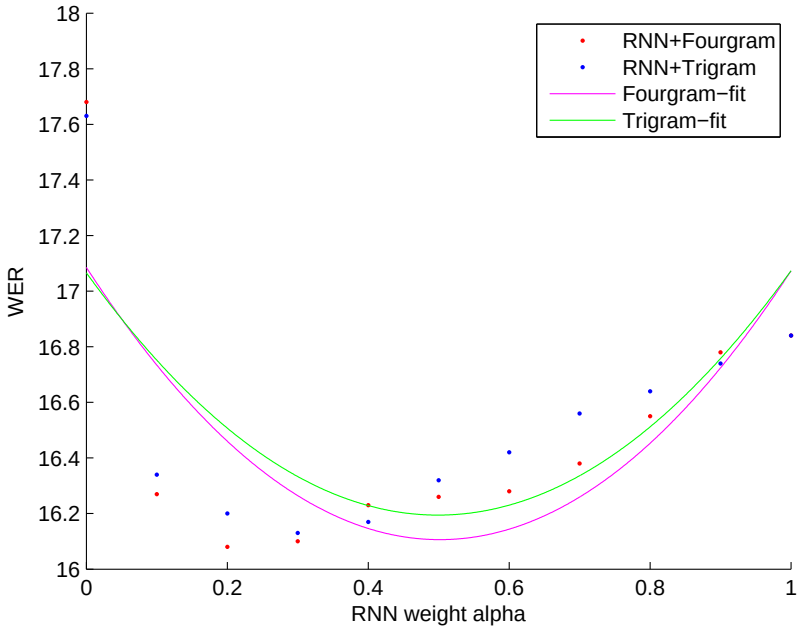

Figure 3.3: RNN+Trigram, RNN+Fourgram Corresponding fits for Dev93

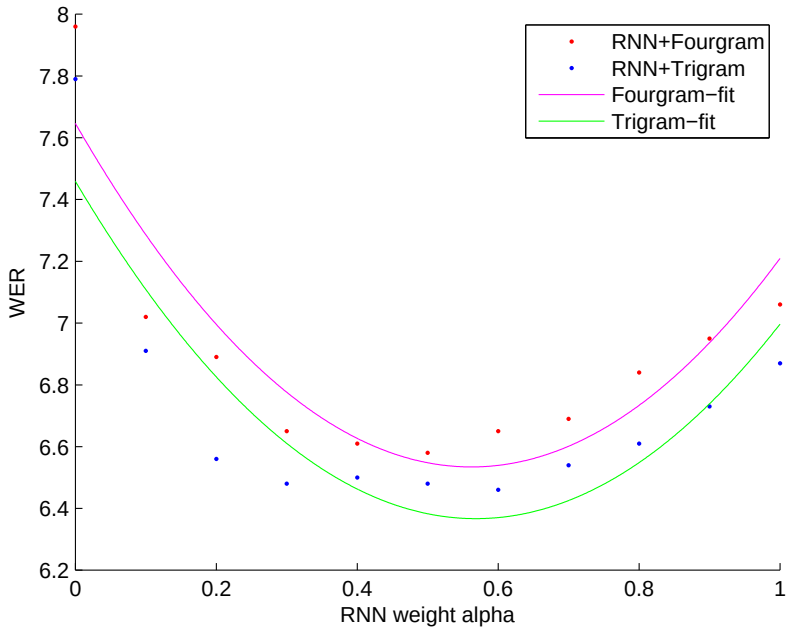

Figure 3.4: RNN+Trigram, RNN+Fourgram Corresponding fits for Eval92

for $\alpha \approx 0.58$. Though the curves are different for tri-gram and four-gram the maximum accuracy point occured for the same α.

This shows that for dev93 data set N-grams perform equally better compared to RNN whereas RNN outperforms N-grams for eval92.

CHAPTER 4

FUTURE WORK

4.1 LSTM based language models

LSTM (Long short term memory) is a type of recurrent neural network. These are best suited to classify, process and predict information even with very long time lags. In a sense these can learn and store the context for a very long time.

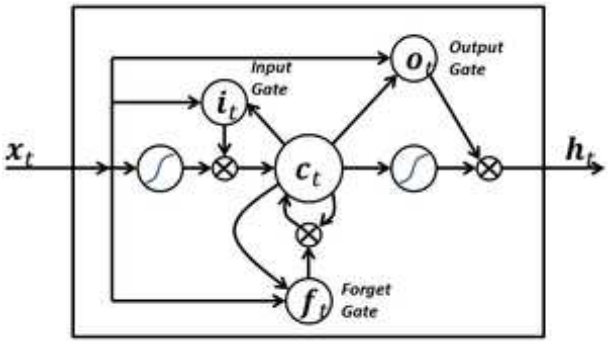

Figure 4.1: A single cell in LSTM network

Unlike RNN's, LSTM's contain an input gate an output gate a forget gate through which the information passes. This complex architecture helps them to train and learn the patterns faster and retain them longer.

The main difficulty in training the RNN's has been the vanishing gradient problem which means that the gradient that is propagated back through the network either decays or grows exponentially. LSTM's architecture is is modified to eliminate the vanishing gradient problem.

4.2 Word2vec

Word2vec is an efficient estimation of the continuous bag-of-words and skip-gram architectures for computing vector representation of words.

Instead of using neural network language models to calculate actual probabilities, one can use the distributed representation encoded in the networks' hidden layers as the representation of words; each word can be mapped onto an n-dimensional real vector called the word embedding, where n is the size of the layer just before the output layer. The representations in skip-gram models have the distinct characteristic that they model semantic relations between words as linear combinations, capturing a form of compositionality. For example, in some such models, if v is the function that maps a word w to its n-d vector representation, then

$$v(\text{king}) - v(\text{male}) + v(\text{female}) \approx v(\text{queen})$$

where $\approx$ is made precise by stipulating that its right-hand side must be the nearest neighbor of the value of the left-hand side.

REFERENCES

[1] Tomas Mikolov.Statistical language models based on neural networks.

[2] Tomas Mikolov, Martin Karafiat, Lucas Burget, Jan "Hanza" Cernocky, Sanjeev Khudanpur. Recurrent neural network based language model (2010).

[3] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan "Honza" Cernocky, Sanjeev Khudanpur. Extentions of recurrent neural network language model.

[4] Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukas Burget, Jan "Honza" Cernocky. RNNLM - Recurrent Neural Network Language Modeling Toolkit.

[5] Stefan Kombrink, Tomas Mikolov, Martin Karafiat, Lukas Burget. Recurrent Neural Network based Language Modeling in Meeting Recognition

[6] Slava M Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer.

[7] I.J.Good. The population frequencies of species and the estimation of popular parameters.

[8] Martin Sundermeyer, Ralf Schluter, and Hermann Ney. LSTM Neural Networks for Language Modeling.