Aalto University

ELEC-E5510

Speech Recognition

Neural Network Language Models
Final Report

Group 3:
Chauhan Rohan
Dubois Antoine

Supervisor:
Gangireddy Siva Reddy

May 15, 2018

Aalto University
School of Science

Abstract

This project work was realized in the context of the course Speech Recognition at Aalto University. The topic of our project was to evaluate the performance of artificial neural networks at language modeling by comparing it to the n-grams method. We therefore first ran trough several scientific papers explaining what language modeling consists in and what where the different methods to approach this problem, including different types of n-grams and neural networks. Then, we set up a series of experiments for n-grams and neural networks and produce results on a given data set. This report presents a review of the literature we read, the setting up of the experiments and their results.

1 Introduction

1.1 Language Models

The goal of statistical language modeling is to learn the joint probability function of sequences of words in a language as specified in both [1] and [2]. In speech recognition, this allows to differentiate between sentence that sounds phonetically similar. Practically, this model can also be represented by the conditional probability of the next word given all the previous ones, since:

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

where w_t is the t-th word of the word sequence, and $w_i^j = (w_i, w_{i+1}, \cdots, w_{j-1}, w_j)$ is the subsequence from the i-th word to the j-th one.

Language modeling offers a range of challenges including the curse of dimensionality, long-distance dependencies of language or how to manage never-seen-before words and a series of algorithm have been proposed to solve those problems. In this review, we focus on the two main approaches, which are n-grams and neural networks, and their variants

To compare those different models, one of the main measure used is perplexity. The idea behind perplexity is to measure how well a probability distribution (or model) will predict unseen data, by determining how many different possibilities there are for a new word. Perplexity is therefore computed as the exponential average of log probability of words in the test data and the goal is to minimize this measure. Perplexity is actually not a really good measure for training algorithm except if the training data is very similar to the test data but it is an interesting approximation.

1.2 N-grams

1.2.1 Basic concept

N-grams is one of the first and most common language models. Those models are based on the Markov approximation that the conditional probability of a word given a sequence of word can be estimated by using only the k last words of this sequence or:

$$P(w_t|w_1^{t-1}) \sim P(w_t|w_{t-k}^{t-1})$$

This approximation leads to a series of models depending on the value of k (e.g for k=0, unigram model, for k=1 bigram model, for k=2 trigram model). The most common of those models is the trigram model and its probabilities can be evaluated using a simple counting rule and a division:

$$P(w_t|w_{t-1}, w_{t-2}) \sim \frac{C(w_{t-2}w_{t-1}w_t)}{C(w_{t-2}w_{t-1})}$$

where $C(w_{t-2}w_{t-1}w_t)$ and $C(w_{t-2}w_{t-1})$ represents respectively the number of occurrences of $w_{t-2}w_{t-1}w_t$ and $w_{t-2}w_{t-1}$ in the training corpus.

Those models alleviate the problem of dimensionality but suffer of the long-distance dependencies of languages which they don't capture.

1.2.2 First ameliorations

These simple n-grams models suffer from a lot of defaults and we list some of the ameliorations that have been proposed to solve those.

A first problem arises when some of the trigrams of the test set never occur in the training data. Indeed, in that case, the previous probability is 0 and one cannot compute the perplexity of the model. The techniques to solve this problem are called smoothing techniques and consist in distributing some probability mass of seen trigrams to unseen trigrams. The most basic of those techniques is the add-1 smoothing which consists simply in adding 1 to the count of each trigram but there exists much more advanced technique than this including the Kneser-Ney smoothing which is one of the state-of-the-art language models.

Secondly, a problem can still occur when the count is different from 0 but very small. Indeed, in that case, the variance of the estimation is very large and is therefore not reliable. A solution to this problem is to rely on the probabilities of the bigrams and unigrams and there are two approaches to do this. The first approach is called back-off and consists in using only the bigrams or unigrams probabilities instead of the trigrams probabilities. The second approach which is a little more sophisticated is called interpolation and consists in taking a weighted average of those different probabilities. Once again there are a lot of different ways to use this approach going from a simple linear interpolation to more advanced choices of the weights.

Finally, a common approach to solve those two problems at once is to collect all the rare words in the training data under a common 'unknown' token and to add the count of all those words to this token. Then, when an unknown word appears, one uses the probabilities related to this token.

1.2.3 More advanced models

After those simple ameliorations, several other techniques have been introduced to cover other limitations of the n-grams models. We already spoke about Kneser-Ney smoothing which is already a very good model in the previous section but we list a series of other models.

- **High-order n-grams**: Obviously using more context will improve the model at the price of a larger computing time and a increasing need for smoothing.

- **Skipping models**: The idea of this technique is to use different contexts than just the preceding words and can yield better performance if combined with the original model.
- **Clustering and classing**: These models attempt to make use of the similarities between words like days of the week, animals, pronouns, etc.
- **Caching models**: These models make use of the observation that if you use a word, you are likely to use it again.
- **Sentence mixture models**: These models are based on the fact that that sentences can be grouped according to their type and that learning a model for each sentence type can improve the performance of the global model.

Finally, those different techniques can all be mixed together leading sometimes to better performance.

1.3 Neural Networks

1.3.1 Neural network as an alternative to n-grams

As said previously, n-grams models have their limitations, the main one being that they do not encompass the long-distance dependencies that are inherent to languages. Another aspect of language aspect that is clearly absent from n-grams is the similarity between words. Neural networks for language modeling were therefore introduced as an alternative to solve those problems.

A first approach using neural networks was proposed by Bengio et al. in the paper *A Neural Probabilistic Language Model* [3]. To use the idea of similarity between words, this paper proposes to represent words as feature vectors and measure probability based on those vectors. Those vectors are then also used to compute the conditional probabilities of the words without relying on the Markov assumption. Taking the words of the paper, the algorithm is the following:

1. associate with each word in the vocabulary a distributed word feature vector,
2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors and the parameters of that probability function.

Using this technique, Bengio et al. could show that they obtained way better results than with n-grams. However, many variants can be used to increase even further the efficiency such as the energy minimizing network, division into sub-networks or multiple vectors per word representation.

1.3.2 Recurrent networks

As stated in [4], one of the limitations of the previous approach is the fact that the algorithm uses a fixed-size context that needs to be specified before learning. A solution to this problem could be to use simple recurrent neural networks. This network takes as input a vector $x(t)$ formed by concatenating vector w representing previous word, and output from neurons in hidden layer at time $t-1$. The output layer $y(t)$ then represents the probability distribution of next word given previous word $w(t-1)$ and the context $h(t-1)$. Therefore, using such an architecture, one can increase the size of the context as much as needed, ideally an infinite context.

1.3.3 LSTMs

As stated, RNNs allows us to increase the size of the context and therefore to theoretically connect previous information to the current task. However, in practice, this is not the case. As the length of the sequene incereases it is difficult to learn the simple RNNs, due to vanishing and exploding gradients problem.

According to [5], LSTMs (Long Short Term Memory networks) are a special kind of RNNs that allows to learn the long distance dependencies without vanishing and exploding gradient problem. In LSTMs, the hidden layer of RNNs is modified such that the new layer consists of various gates (input, output and forget) and a cell state, each having a special role and interesting in a specific way to let the information from previous modules to go through the network and therefore being better at learning the long distance dependencies.

The best results obtained in speech language modeling using RNNs are actually obtained using LSTMs. However, there is still more to explore and GRUs or RNNs in generative models seem very promising tracks to explore.

2 Experiments

2.1 Data set

The data set we used for our experiments is the Penn Treebank data set [6]. This data set is made up of more than 1,000,000 words taken mainly from the 1989 Wall Street Journal with about 10,000 different unigrams. This data set was divided into three parts:

- Training set: $\sim 890,000$ words
- Validation set: $\sim 70,000$ words
- Test set: $\sim 80,000$ words

Using this data set, we set up experiments both for the n-grams and artificial neural networks method using different configurations. The evaluation metric for all these models is perplexity.

2.2 N-grams

Let's first state that for this part of the experiments, the validation set was actually just used as another test set. Indeed, we trained all the following models on the training set for sizes of context going from 3 to 10. Then we tested the trained models on the validation set and test set separately instead of selecting the best parameter using the validation set and then retraining a bigger model on the training and validation set and computing the perplexity on the test set. We will thus refer to those two as the test sets for this part.

The tools that we used to compute the different n-grams models is SRILM - The SRI Language Modeling Toolkit [7]. In particular, we made use of the functions *ngram-count* to train the models and *ngram* for the testing.

In n-grams, the state-of-the-art is the Kneser-Ney smoothing which normally combines both smoothing and interpolation techniques. We decided therefore to test the following possibilities.

First, we tried the basic n-grams method where no smoothing or interpolation is used. In that case, probabilities are just computed as the division between the counts of n-grams divided by the number of n-1 grams.

Then, having use the most basic technique, we tried the most advanced by trying the Kneser-Ney smoothing method with interpolation. We also wanted to see the effect of interpolation in this method so we tried Kneser-Ney smoothing by disabling interpolation.

Finally, we wanted to compared those results to two smoothing techniques. Firstly, a technique called Witten-Bell smoothing which is a smoothing method that transfers some probability mass to the unseen n-grams by redefining the way the n-grams are counted. Secondly, we tried the Ristad's natural discounting law.

For all this models, we are also using the unknown-word token as a regular word.

2.3 Artificial neural networks

For language modeling using Neural Networks, we used different type of layers, different width, namely 128 and 256, and depth, 1 or 2, and compared their results keeping the hyper parameters same. The architecture consisted of a initial projection layer having size of 100 and taking word input. The output from the projection was fed into hidden layer of variable type. The hidden layer served as the input for final layer which consisted of a soft-max unit. One model was also trained with two hidden layers. After the model was trained using training data and validation data, it was tested on test data.

The optimization method used in training was adagrad with L_2 regularization of 0.00001. Cross-entropy was taken as the cost function and a learning rate of 1. The stopping criterion was when no further improvement could be done by training. The patience for neural network was taken as 0 with a validation frequency of 1 and max gradient norm as 5.

The different types of hidden layer used in the models were Tanh, GRU, BGRU and LSTM (for which we tried with 2 hidden layers). The tanh layer has tanh as activation function. Tanh has characteristics similar to sigmoid function but has a more stronger gradient. GRU stands for gated recurrent unit and BGRU stands for bidirectional gated recurrent unit. LSTM stands for long short term memory. GRU and LSTM utilize different way to solve vanishing gradient problem. GRU doesn't use a memory unit and exposes full hidden content without any control.

The tools used to train and build these models were theanolm [8] and SRILM tool [7]. The environment was setup using Anaconda.

3 Results

3.1 N-grams

The results for the n-grams experiments are displayed in Figures 1 and 2. As the two test sets are taken from the same corpus, it can be assumed that the data they contain is distributed according to the same model and that the results should therefore be quite similar. If we accept the fact that the perplexity on the validation set is constantly higher than on the test set, which is probably due to existence of rarer n-grams in the validation set, we see that indeed the

perplexity for each model evolves quite similarly between the two sets for increased context size.

We will start by analyzing the general trend across all the models. As we can see on the figures, for all the models except add-one smoothing, there is a big drop in perplexity when going from 3-grams to 4-grams and a slightly smaller but considerable going from 4-grams to 5-grams. From 5-gram to 9-grams then, the perplexity tends to stabilize for all models except for the KN smoothing without interpolation which is starting to increase and still the add-one smoothing. Finally, we have a very strange behavior going to 10-grams as the models without interpolations increase quite consequently while the models using interpolation decreases again.

From this analyzes, we can see that interpolation is primordial to prevent perplexity to rise when increasing the context size. This can easily be understand by the fact that for larger n-grams, one encounters more and more OOV n-grams and therefore we must rely more and more on interpolation. In addition, we can see that going above 5-grams does not make a lot of sense. Indeed, the decrease in perplexity is nearly non-existent while the time complexity is increasing more and more. A priori, we would actually expect the perplexity not to decrease suddenly again for 10-grams but this can maybe be explained by the sizes of the training set and test sets which are actually really small.

If we now compare the different methods with each other, the results are actually quite surprising. Indeed, as expected, the Kneser-Ney smoothing outperforms all other methods even without interpolation, even though in that case, the perplexity is already much higher. However, the results for all the other smoothing methods are quite unexpected as none of them outperforms the no-smoothing method. This is probably explainable by the fact that we are training the models on very small data sets and that the disadvantages of those smoothing techniques take over their advantages.

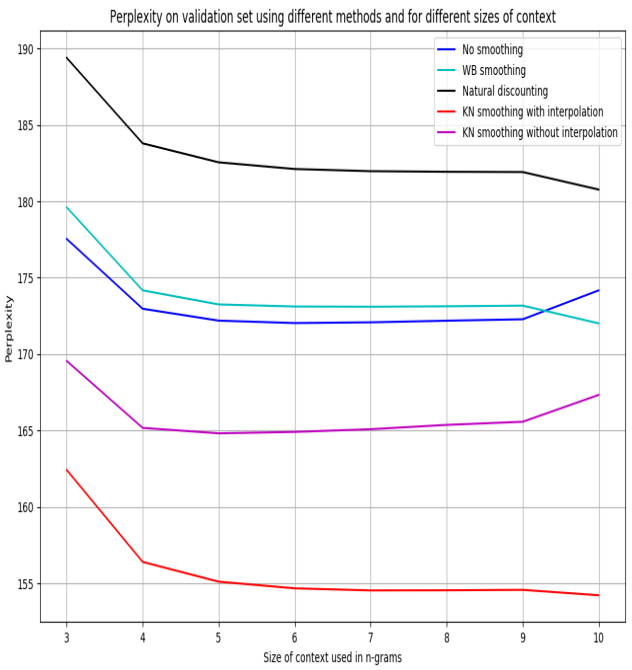

Figure 1: Perplexities for the validation set without add-one smoothing

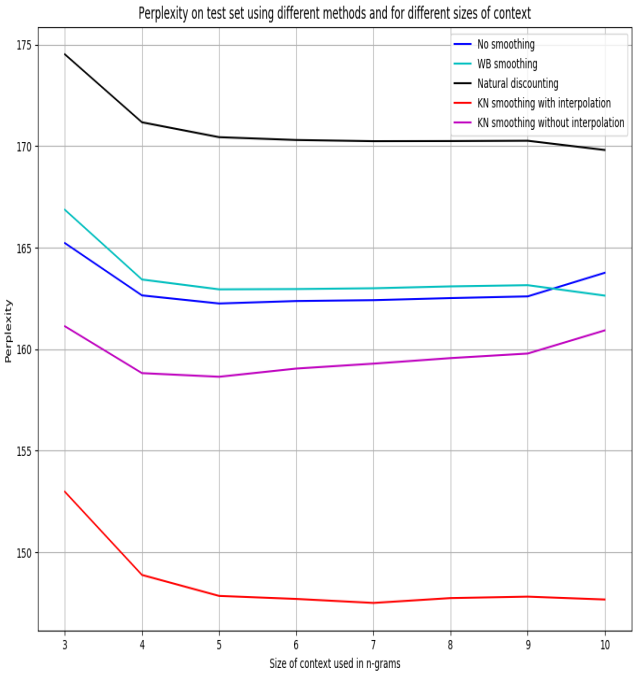

Figure 2: Perplexities for the validation set without add-one smoothing

3.2 Artificial neural networks

Figure 3 and Table 1 show the results for the validation and test perplexity respectively. Figure 3 is mainly there to illustrate how the quality of the different models evolves with the number of epochs and how for each model, this performance start to stabilize after a certain number of epochs.

Then looking at either the graph or the table we can infer similar observations. As we can observe, BGRU hidden layers performs the best and Tanh layers perform the worst. In addition, the performance of GRU is on par with LSTM. Finally, we see that adding a second layer of LSTM seems to have improve the performance compared to the one layer model.

Moreover, if we compare the perplexities for validation and test set, we see some differences. In particular, we see that the influence of the width of the model is not the same. For validation set, the perplexity decreases when the width increases but for the test set, it is not always the case. Indeed, for two types of hidden layers, the perplexity on the test set increases while for the two other types it decreases it. This can be probably be explained once again because of the size of the network. As we try to train more complex model on a small data set, over-fitting appears and this increase the generalization error.

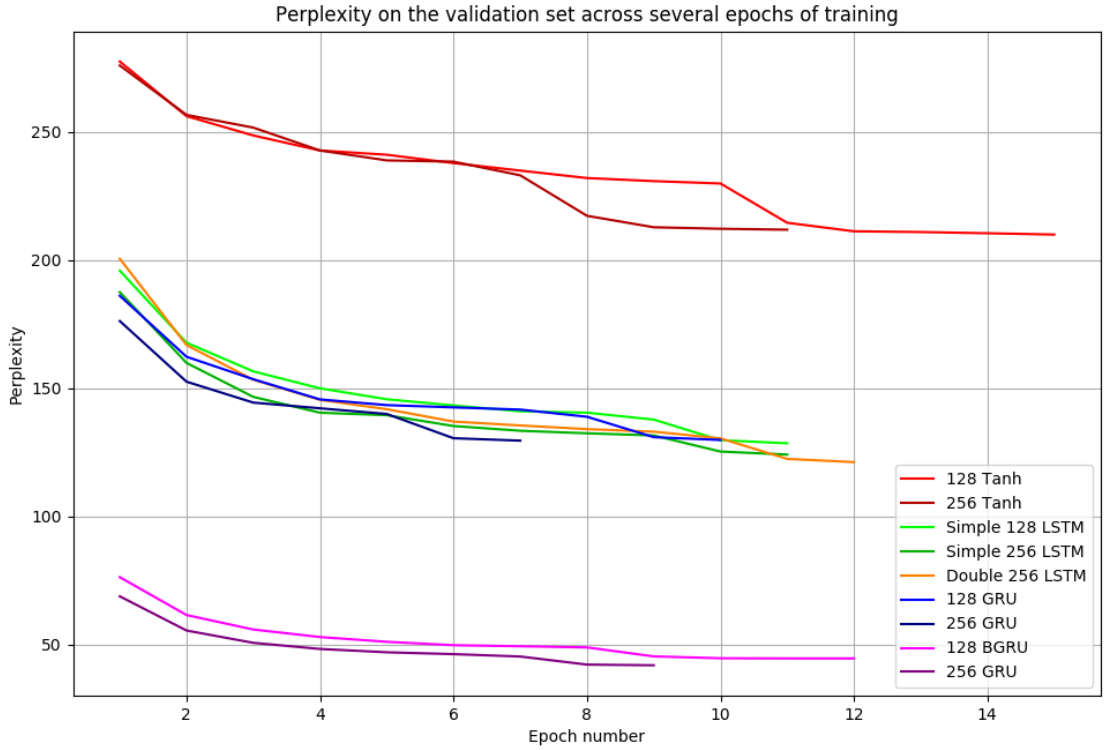

Figure 3: Validation perplexity of different neural networks during training of the model

Hidden Layer Type	Depth	Width	Perplexity
Tanh	1	128	173.74
		256	174.85
LSTM	1	128	108.17
		256	105.73
	2	256	102.83
GRU	1	128	109.79
		256	110.84
BGRU	1	256	36.97
		128	34.82

Table 1: Test perplexity for different neural networks

3.3 Comparison between n-grams and NNs

Having looked at the results for the two main techniques we were analyzing separately, we want to finish this report by comparing those. Figure 4 shows such a comparison. Note that for n-grams models, we systematically took the best size of context.

The results follows what was suggested in the articles. Indeed, all the neural networks, except for the tanh layer, outperform the state-of-the-art technique of n-grams, Kneyser-Ney smoothing.

However, it is important to make the following note to mitigate the results. Indeed, in this report, we did not speak about time complexity which is a main downside of neural networks. Indeed, compared to n-grams which are trained and tested in a few seconds, neural networks trained on CPU take several hours to train and a few minutes to test. On GPUs, the time performance are better but still way longer than for n-grams.

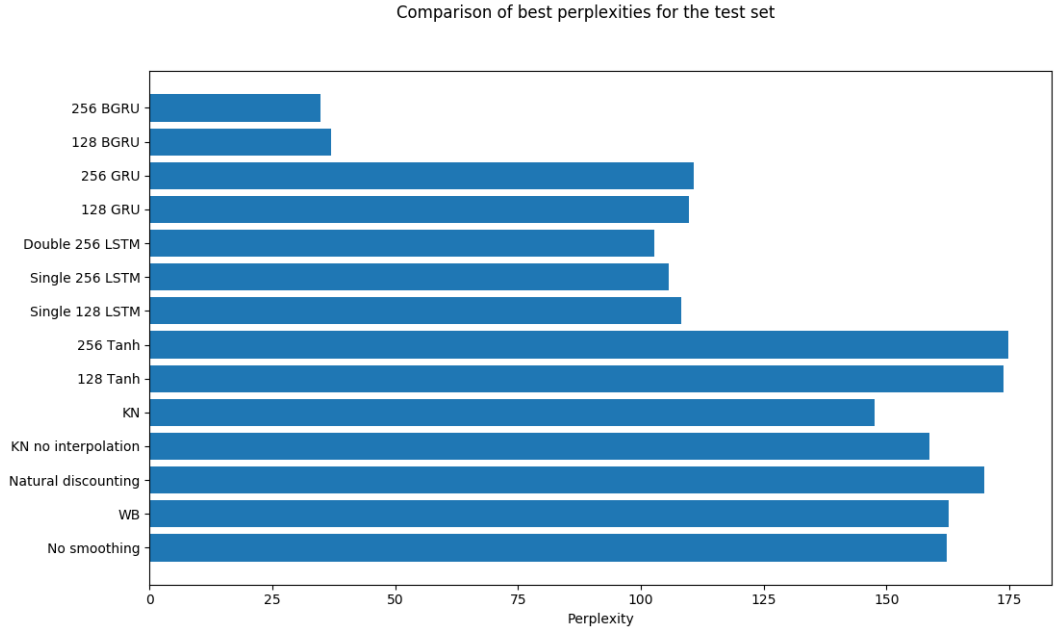

Figure 4: Comparison between N-grams and Neural networks for language modeling

4 Conclusion

Throughout this project, we had the opportunity to deepen our knowledge about language modeling by testing different techniques to solve this problem. Given the results we obtain at the end for those different techniques, we were able to confirm what was stated in the different articles we read while noting some deviations compared to the expected results due to limitations in the data.

References

[1] Joshua T. Goodman, *A Bit of Progress in Language Modeling, Extended Version*, Machine Learning and Applied Statistics Group, Microsoft Research

[2] Professor Dan Jurafsky and Chris Manning, *Introduction to N-grams*, Stanford NLP, https://www.youtube.com/watch?v=TPE9uSFFxrI

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin, *A Neural Probabilistic Language Model*, Département d'Informatique et Recherche Opérationnelle, Centre de Recherche Mathématiques, Université de Montréal

[4] Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan "Honza" Cernocky, Sanjeev Khudanpur, *Recurrent neural network based language model*, Speech@FIT, Brno University of Technology, Czech Republic and Department of Electrical and Computer Engineering, Johns Hopkins University, USA

[5] Christopher Olah *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] *Treebank-3*, Linguistic Data Consortium, University of Pennsylvania, https://catalog.ldc.upenn.edu/ldc99t42

[7] *SRILM - The SRI Language Modeling Toolkit*, http://www.speech.sri.com/projects/srilm/

[8] Seppo Enarvi and Mikko Kurimo, *TheanoLM - An Extensible Toolkit for Neural Network Language Modeling*, http://www.speech.sri.com/projects/srilm/, CoRR, 2016