Artificial Neural Networks: Can They Learn Language Models?

Ashiq Sazid
Ayan Roy
Hk Mehedi
Md.Farhadul Islam
Annajiat Alim Rasel
School of Compueter Science,BRAC UNIVERSITY
Dhaka

Abstract

The most popular and widely applied models for statistical language modeling at the moment are N-gram models. In this paper, we looked into a different approach to creating language models, namely,

language model learning with artificial neural networks. The outcome of our experiment demonstrates that a neural network is capable of learning a language model that performs even better than conventional statistical techniques.

1 Introduction

Speech recognition, text classification, optical character recognition, etc. all use language models in some capacity. Another effective method that is widely applied in many areas of computer science is artificial neural networks (NN). A strange phenomenon is that, despite the popularity of artificial neural networks, it is difficult to locate any literature on language modeling using NN, despite the fact that there are some works on connectionist natural language processing (e.g. [7]). The lack of research into using NN for language modeling may be due to two factors. The first is that it is logical to assume that the traditional statistical approach is better suited to solve this issue. The second is that the neural network's size required for this. The second is that the size of the neural network required to solve this issue is excessive, and training would proceed at an intolerably slow pace. In this study, we conducted NN experiments to create language models and looked into issues unique to using neural networks for language modeling.

2 Basic of language modeling

Language model is used to assign a probability $P(\mathbf{W})$ to every possible word sequence W. Using Bayes rule of probability, $P(\mathbf{W})$ can be decomposed as

$$P(\mathbf{W})=\prod_{t=1}^{n} P\left(w_{t} \mid w_{1}, \wedge, w_{t-1}\right)=\prod_{t=1}^{n} P\left(w_{t} \mid \mathbf{h}_{t}\right)$$

where $\mathbf{h}_t$ denote the history of word $w_t, w_1, \wedge, w_{t-1}$. So the task of language model is to estimate the probability of a word given its history. Because there are a huge number of different histories, it is impractical to specify all $P\left(w_t \mid \mathbf{h}_t\right)$ completely. If we can map the histories into some number of equivalence classes, and let Φ be this mapping. Then the conditional probability can be approximated by

$$P\left(w_{t} \mid \mathbf{h}_{t}\right)=P\left(w_{t} \mid \Phi\left(\mathbf{h}_{t}\right)\right)$$

A commonly used equivalence classification is to consider the histories that end with same n $-$ 1 words as one equivalent class. For $n=2$, we get bigram language model:

$$P(\mathbf{W})=\prod_{t=1}^{n} P\left(w_{t} \mid w_{t-1}\right)$$

In this paper, we will focus on bigram model. The estimation of the probability $P\left(w_2 \mid w_1\right)$ is straightforward:

$$P\left(w_{2} \mid w_{1}\right)=f\left(w_{2} \mid w_{1}\right)=\frac{\operatorname{Count}\left(w_{1}, w_{2}\right)}{\operatorname{Count}\left(w_{1}\right)}$$

where $f\left(w_2 \mid w_1\right)$ denotes the relative frequency derived from data. For the reason of data sparsity, it is necessary to smooth the bigram frequencies. One simple way to do this is to interpolate bigram, unigram, and uniform frequencies:

$$P\left(w_{2} \mid w_{1}\right)=\lambda_{2} f\left(w_{2} \mid w_{1}\right)+\lambda f\left(w_{2}\right)+\lambda f_{0}$$

Where f_0 is the uniform probability and equal to $1/|V|(|V|$ is the vocabulary size). λ, λ and λ_2 satisfy $\lambda_2+\lambda+\lambda_2=1$. The interpolation weights can be estimated using EM algorithm.

The measure of the performance of a language model usually is perplexity. The perplexity of a language model over a particular corpus is the inverse of the average probability per word calculated by the language model:

$$\text{Perplexity }_{M}=\left(P_{LM}(\mathbf{W})\right)^{\frac{1}{n}}$$

where n is the size of the corpus.

The lower the perplexity of language model, the better the language model is.

3 Neural Networks Approach

3.1 Input and output encoding

In our experiments, the network has $|V|$ input units and $|V|$ output units, where $|V|$ is the vocabulary size. The i^{th} input unit is 1 if the current word is w_i. The value of i^{th} output unit represents the probability of W_i being the next word.

3.2 Error function

Because our goal is to minimize the perplexity, so we use the logarithm of perplexity as error function. This is same as the negative log likelihood.

$$E = -\sum_t \log o_{t,w_t}$$

When training the neural network, we need to minimize the above error function. It can be proven that using this error function the value of i^{th} output will converge to $P(w_i \mid w_j)$ if the input to the network is word w_j. So upon convergence, the network will be equivalent to a bigram language model without any smoothing. Without smoothing, the performance on test data will be very poor, so methods of preventing overfitting will be very important to us. In this paper, we use early-stopping to prevent overfitting, i.e. stop the training when the network achieves best performance on holdout data.

3.3 Activation function

We use softmax activation function [3] for the output units, which guarantee the sum of the outputs is 1 . Let the net input to each output unit be y_i, then the softmax output O_i is:

$$o_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

where the net input is $y_i = w_{i,o} + \sum_j w_{ij} x_{ij}$
and X_{ij} is the j^{th} input to this unit.

3.4 Network structure

In this paper we only experiment with single layer network. The input units and output units are fully connected, so we have $|V| \times (||+1)$ weights (including bias weight).

3.5 An issue on computation cost

A major problem for training such a neural network to learn language model is that it is very computationally expensive. So we should try all possible ways to reduce the computational cost. One important characteristic for the neural network of our problem is the sparsity of its inputs (i.e. most of them are zero). Using this fact, and notice the formula for updating weights in back-propagation algorithm:

$$\Delta w_{ij} = \mathbf{nQ}_1 x_{ij}$$

So the weight will not be changed if the corresponding input value is zero. Thus we can save a vast amount of computation if we only update those weights with non-zero input value.

3.6 Training method

To train our network, we employ the back-propagation algorithm. The starting weights are zero (This is equivalent to an uniform distribution). Utilizing batch training (i.e. update weights after a whole epoch). As was mentioned in section 3.2, it is crucial that we avoid overfitting in this situation. To avoid overfitting, we select a modest constant learning rate and employ early stopping. When the holdout set's perplexity falls to its lowest level, the training is terminated.

4 Experiment result

Even with a moderate vocabulary size (such as 10"*mathrmK*"), the size of the network becomes very large because the number of input units and output units of the neural network is determined by vocabulary size. Therefore, for our experiment, we need to use a corpus with a small vocabulary size. The data gathered using the Communicator Telephone Air Travel Information System is the corpus that we use [8]. Communicator's language model is a class-based language model. The vocabulary size is roughly 2500 words.The language model has about 1200 classes, 20 of which correspond to word classes like "city" and "airport," etc., while the remaining classes each correspond to a single word. The within-class probability is not taken into account in the perplexities reported in this paper.

We contrast the neural network's performance with that of three language model smoothing methods. Table 1 displays the outcome. The most popular smoothing techniques are Katz and Jelinek-Mercer smoothing [1][5]. The most well-known smoothing technique is Kneser-Ney [2][6].

Method	Training	Holdout	Test	Test (no 0ov)
Katz	9.84	11.89	12.20	10.26
Jelinek-Mercer	9.50	11.76	11.99	
Kneser-ney	9.58	11.19	11.17	9.68
NN	9.82	11.24	11.16	9.58

Table 1 Performance of standard methods

5 Discussion

The outcomes of the experiment demonstrate that neural networks are capable of learning language models that perform on par with conventional techniques. Additionally, our findings show that neural networks perform even better than traditional statistical approaches. Given that we did not explicitly smooth neural networks, this fact is unexpected. Early-stopping is the only rule used to stop overfitting.

Although neural networks perform better than conventional methods on perplexity, their computational cost is much higher. The Typical training time for one epoch is about one minute on a Pentium III 500MHz machine and we need thousands of epochs for training.On the other hand, we can obtain a Katz or Jelinek-Mercer language model in less than 30 seconds.

We look into how precisely early-stopping could provide a model with good generalizability on test data. To start, we attempt to analyze the following straightforward network:

- There is no hidden unit in the network.
- There is no bias weight.
- The output is the linear summation of input units.
- Use batch updating and the learning rate is small enough.
- The target value of j$^{\text{th}}$ output unit is set to 1 and the other output units are set to 0 when the network is presented with a word pair (i, j).
- Use squared error as error function. Based on above simplified assumption, we can derive how weights evolve during training (see Appendix for details):

$$w_{ij}(t) = \left(1 - e^{-\text{Count }(i)t}\right) f_{ij} + e^{-\text{Count }(i)t} w_{ij}(0)$$

6 Where

Count (i, j) is the count of bigram (i, j)

Count (i) is the count of word j

N is the size of corpus

$f_i = \text{Count}(i)/N$

$f_{ij} = \text{Count}(i, j)/\text{ Counti})$

Notice that $w_{ij} \to f_{ij}$ when $t \to \infty$, which is what we expected. So when early stopping occurs, w_{ij} is a weighted sum of two terms, the initial weight and f_{ij}. If the initial weights are acquired from some known distribution D, say the unigram distribution, then the resulting weights are the interpolation of D and the bigram frequency f_{ij} from training corpus and the interpolation weights are determined by Count (i) :

$$w_{ij}(t) = \left(1 - e^{-\text{Count }(i)_i t}\right) f_{ij} + e^{-\text{Count }(i) t} f_j$$

For small Count $(i), w_{ij}$ will be close to D. For large Count $(i), w_{ij}$ will be close to f_{ij}. This seems quite reasonable given that for smaller Count (i), we cannot get a robust estimation for f_{ij}, thus our estimation should depend on the prior knowledge heavier. Above equation is interesting if we compare it with the general form of Jelinek-Mercer smoothing [1]

$$w_{ij}(t) = (1 - \lambda(\text{ Counti } i))) f_{ij} + \lambda(\text{ Counti } i)) f_j$$

where the interpolation weight λ is a function of count.

We need to note here that the smoothing in equation (5.2) does not lead to good performance. In fact, our experiment shows that the perplexities for this simple network when earlystopping occurs are 12.17 for training set, 14.63 for validation set, and 19.24 for test set.

In figure 1(b), we plotted the interpolation weights acquired by the simplified network for different Count (i). Using EM (Expectation-Maximization [4]) algorithm, we calculated the optimal interpolation weight for different Count(i) [1] in our corpus and figure 1 (a) shows λ (Count) for different Count (i).

From figure 1 we can see that when Count (i) is small, the interpolation weight calculated by equation (5.2) is much larger than the optimal interpolation weights; when Count (i) is large, When Count (i) is high, the interpolation weight calculated by equation (5.3) is much smaller than the optimal interpolation weights, resulting in a difference between the interpolation weight calculated by equations (5.2) and (5.3) of a significant size. This suggests that even after overfitting for words with high word counts, words with low word counts may not receive enough training. This is the main cause of the poor performance of the interpolation provided in (5.2). Even though the simple linear output network's smoothing is not satisfactory, it gives us some insight into how the neural network in section 4's performance can be so excellent. In comparison to the network we have just examined, the network used in section 4 is considerably more complex. We do not know the exact form of the interpolation it achieves. However, from above analysis, we can reasonably infer that the neural networks in our experiment are indeed performing some kind of smoothing, which resulted in heavy smoothing for those events infrequent in training data and light smoothing for those events frequent in training data.

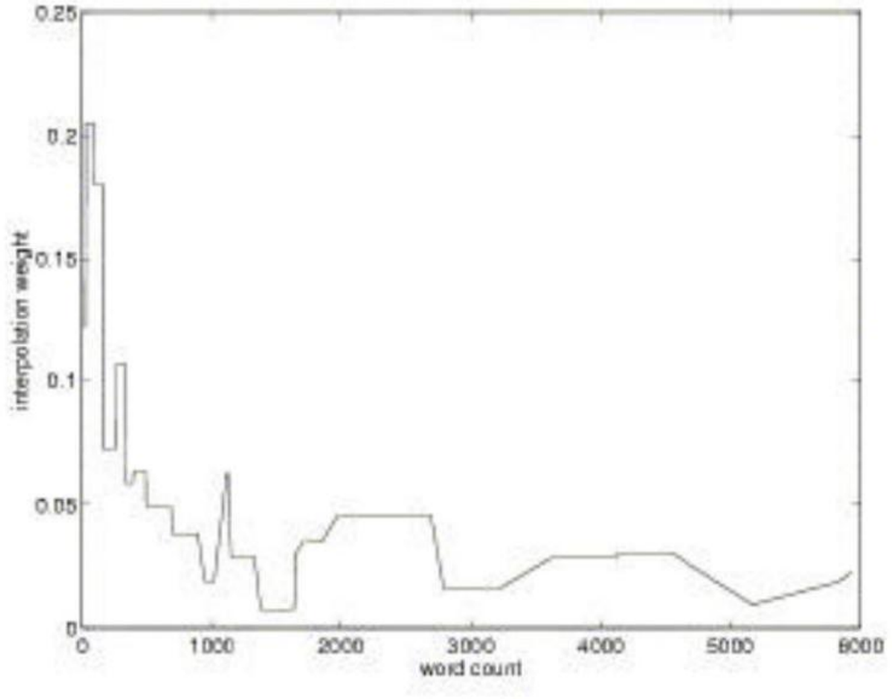

Figure 1(a) optimal weight

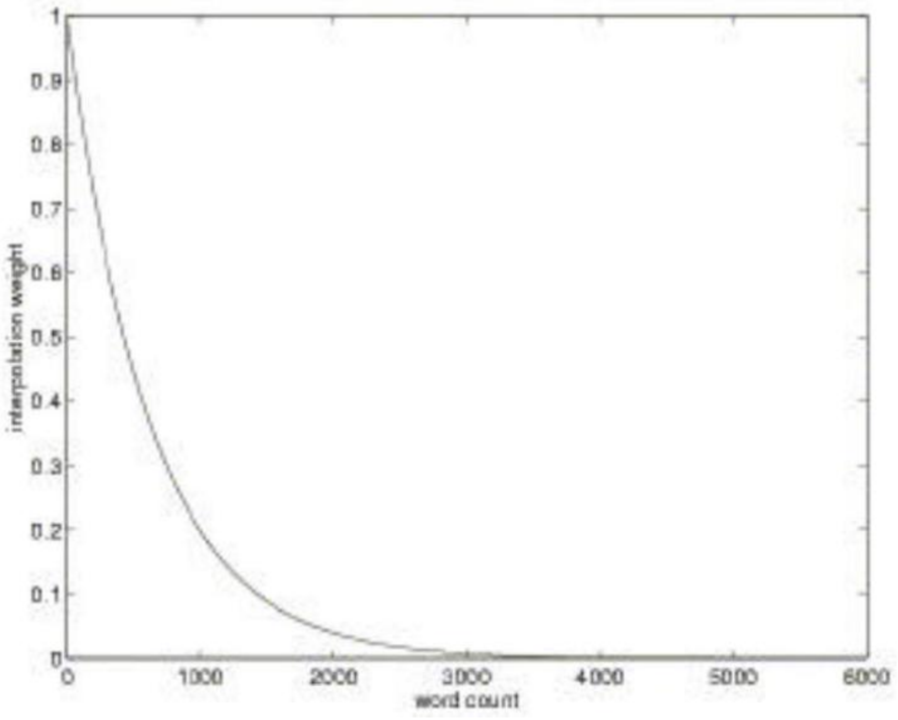

Figure 1(b) weight from equation
Figure 1 interpolation weight vs. word count

7 Conclusions and Future Work

In this work, we try to use an alternative approach to build language models. Our experiment results show that neural networks can learn language models that have performance comparable with other standard methods. However, the computational cost is much higher than standard methods. We have tried to get some understanding on how neural network can have a good performance on language modeling.

There are many things worth to be tried in the future work. For example, experimenting on another domain, adding some hidden units, adding more history to the input or adding recurrent connections so that the network can model the long distance dependencies, etc.

8 Acknowledgement

This research was sponsored by the Space and Naval Warfare Systems Center, San Diego, under Grant No. N66001-99-18905. The content of the information

in this publication does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

9 Appendix

The error for a word pair (i, j) is

$$\frac{1}{2}(w_{ij}-1)^2+\frac{1}{2}\sum_{i\neq j} w_{il}^2 = \frac{1}{2} = 1-2w_{ij}+\sum_{i} w_{il}^2 =$$

The error is for a whole epoch is

$$E=\frac{1}{2}\sum_{i,j}\text{Count}(i,j) = 1-2w_{ij}+\sum_{i} w_{il}^2 =$$

$$=\frac{1}{2}\sum_{l,j}\text{Count}(i,j)-\sum_{i,j}\text{Count}(i,j)_{w_{ij}}$$

$$+\sum_{l}\quad\sum_{j}\text{count}(i,j)\;\; w_{il}^2$$

$$=\frac{N}{2}-\sum_{i,j}\text{Count}(i,j)_{w_{ij}}+\frac{1}{2}\sum_{i,j}\text{Count}(i)_{w_{ij}^2}$$

$$=\frac{N}{2}+\frac{1}{2}\sum_{i,j}\text{Count}(i) = w_{ij}^2-2\frac{\text{Count}(i,j)}{\text{Count}(i)}w_{ij}$$

$$=\frac{N}{2}+\frac{1}{2}\sum_{i,j}\text{Count}(i) = w_{ij}-\frac{\text{Count}(i,j)}{\text{Count}(i)}-\Bigg]^2 = \text{Count}(i,j)|^2 =$$

Where
Count (i, j) is the count of bigram (i, j)
Count (i) is the count of word i
N is the size of corpus
Let f_i and f_{ij} be $f_i = \text{Count}(i)/N$,
f_{ij} = Counti, j) / Counti)
So we have

$$\frac{\partial E}{\partial w_{ij}} = \text{Count}(i)\,(w_{ij}-f_{ij})$$

According to the gradient descent rule, we have

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = -\,\text{Count}(i)\,(w_{ij}-f_{ij})$$

Because η is small enough, we can use following differential equation to describe the evolution of w_{ij}

$$\frac{dw_{ij}}{dt} = -\,\mathrm{Count}(i)\,(w_{ij} - f_{ij})$$

The solution to the above differential equation is

$$w_{ij}(t) = f_{ij} + (w_{ij}(0) - f_{ij})\, e^{-\text{-ount }(i)t}$$

The above equation can be rewritten as

$$w_{ij}(t) = \left(1 - e^{-\,\mathrm{Count}(i)t}\right) f + e^{-\,\mathrm{Count}(i)k} w_{ij}(0)$$