Can Artificial Neural Networks Learn Language Models?

Wei Xu, Alex Rudnicky
School of Compueter Science, Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213, USA
xw@cs.cmu.edu, air@cs.cmu.edu

Abstract

Currently, N-gram models are the most common and widely used models for statistical language modeling. In this paper, we investigated an alternative way to build language models, i.e., using artificial neural networks to learn the language model. Our experiment result shows that the neural network can learn a language model that has performance even better than standard statistical methods.

1. Introduction

Language models are widely used in speech recognition, text classification, optical character recognition, etc. Artificial neural networks (NN) are also a powerful technique that is widely used in various fields of computer science. Though there are some works on connectionist natural language processing (e.g. [7]), a strange phenomenon is that in spite of the popularity of artificial neural networks, it is hard to find any work on language modeling using NN in the literature. There might be two reasons for the lack of work on using NN for language modeling. The first is that it is reasonable for one to think that the standard statistical method is more suitable for this problem. The second is that the size of the neural network needed for this problem is too huge and the training would be too slow to be tolerable.

In this paper, we did experiments of using NN to build language model and investigated issues specific to applying neural networks to language modeling.

2. Basic of language modeling

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

$$P(\mathbf{W}) = \prod_{t=1}^{n} P(w_t \mid w_1, \Lambda\ , w_{t-1}) = \prod_{t=1}^{n} P(w_t \mid \mathbf{h}_t) \quad (2.1)$$

where $\mathbf{h}_t$ denote the history of word w_t, $w_1, \Lambda\ , w_{t-1}$. 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 $P(w_t \mid \mathbf{h}_t)$ 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

$$P(w_t \mid \mathbf{h}_t) = P(w_t \mid \Phi(\mathbf{h}_t)) \quad (2.2)$$

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:

$$P(\mathbf{W}) = \prod_{t=1}^{n} P(w_t \mid w_{t-1}) \quad (2.3)$$

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

$$P(w_2 \mid w_1) = f(w_2 \mid w_1) = \frac{Count(w_1, w_2)}{Count(w_1)} \quad (2.4)$$

where $f(w_2 \mid w_1)$ 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:

$$P(w_2 \mid w_1) = \lambda_2 f(w_2 \mid w_1) + \lambda_1 f(w_2) + \lambda_1 f_0 \quad (25)$$

Where f_0 is the uniform probability and equal to $1/|V|$ ($|V|$ is the vocabulary size). λ_0, λ_1 and λ_2 satisfy $\lambda_0 + \lambda_1 + \lambda_2 = 1$. 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:

$$Perplexity_{LM} = (P_{LM}(\mathbf{W}))^{-\frac{1}{n}} \quad (2.6)$$

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.

$$E = -\sum_t \log o_{t,w_t} \quad (3.1)$$

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:

$$o_i = \frac{e^{y_i}}{\sum_j e^{y_j}} \quad (3.2)$$

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 $|V| \times (|V| + 1)$ 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:

$$\Delta w_{ij} = \eta \delta_i x_{ij} \quad (3.3)$$

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

We use back-propagation algorithm to train our network. The initial weigts are set to zero (This is equivalent to an uniform distribution). We use batch training (i.e. update weights after a whole epoch). As mentioned in section 3.2, it is particularly important for us to prevent overfitting in this problem. We choose a small constant learning rate and use early stopping to prevent overfitting. The training is stopped when the perplexity on the holdout set reaches the lowest point.

4. Experiment result

Because the number of the input units and output units of the neural network is the vocabulary size, the size of the network becomes very huge even with a moderate vocabulary size (e.g. 10K). So we need to use a corpus with relatively small vocabulary size for our experiment. The corpus we use is the data collected using the Communicator Telephone Air Travel Information System [8]. The language model for Communicator is a class based language model. The total vocabulary size is about 2500. There are about 1200 classes in the language model, among which 20 classes correspond to word classes such as [city] and [airport], etc, while each of the remaining classes corresponds to a single word. The perplexities reported in this paper do not take the within-class probability into account.

We compare the performance of the neural network with 3 language model smoothing techniques. Table 1 shows the result. Katz and Jelinek-Mercer smoothing [1][5] are the most widely used smoothing methods. Kneser-Ney is the best-known smoothing method [2][6].

Method	Training	Holdout	Test	Test (no oov)
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 experiment results show that neural networks can learn language models that have performance comparable with standard methods. And in our result, the performance of neural network is even better that that of the standard statistical methods. This fact is surprising in that we did not use any explicit smoothing in neural networks. The *only* rule used to prevent overfitting is early-stopping.

Though the performance on perplexity is encouraging, the computational cost for neural networks is much higher than the standard methods. 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 only need less than half a minute to get a Katz or Jelinek-Mercer language model.

We try to understand how exactly early-stopping can possibly give us a model that has good generalization ability on test data. To gain the insight, we first try to analyze following simple 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[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):

$$w_{ij}(t) = \left(1 - e^{-Count(i)t}\right) f_{ij} + e^{-Count(i)t} w_{ij}(0) \qquad (5.1)$$

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

$f_i = Count(i)/N$,

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

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 $Count(i)$:

$$w_{ij}(t) = \left(1 - e^{-Count(i),t}\right) f_{ij} + e^{-Count(i)t} f_j \qquad (5.2)$$

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].

$$w_{ij}(t) = (1 - \lambda(Count(i)))\, f_{ij} + \lambda(Count(i))\, f_j \qquad (5.3)$$

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 early-stopping 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 $\lambda(Count)$ 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, the interpolation weight calculated by equation (5.3) is much smaller than the optimal interpolation weights. This suggests that when overfitting occurs for the words with large count, those words with small count still do not get enough training. This is the reason why the interpolation given by (5.2) cannot get a good performance.

Though smoothing for the simple linear output network is not satisfactory, it provides us some insight on how the neural network in section 4 can have very good performance. The network used in section 4 is much more complicated than the network we have analyzed above. 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.

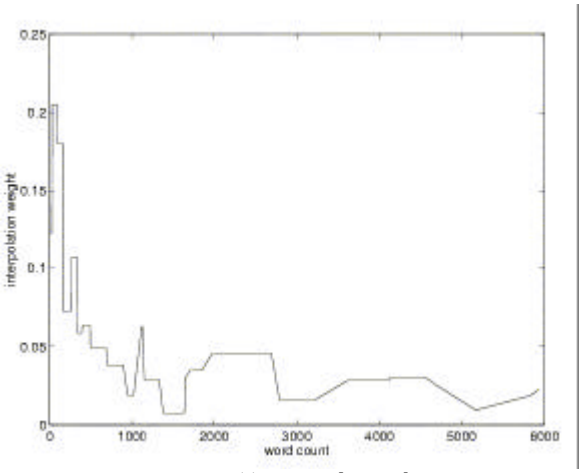

Figure 1(a) optimal weight

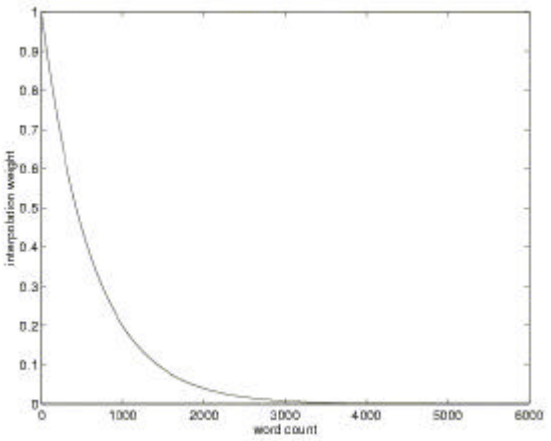

Figure 1(b) weight from equation

Figure 1 interpolation weight vs. word count

6. 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.

Acknowledgement

This research was sponsored by the Space and Naval Warfare Systems Center, San Diego, under Grant No. N66001-99-1-8905. 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.

Appendix

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

$$\frac{1}{2}(w_{ij}-1)^2+\frac{1}{2}\sum_{l\neq j}w_{il}^2=\frac{1}{2}\left(1-2w_{ij}+\sum_l w_{il}^2\right)$$

The error is for a whole epoch is

$$\begin{aligned}
E &= \frac{1}{2}\sum_{i,j}Count(i,j)\left(1-2w_{ij}+\sum_l w_{il}^2\right)\\
&= \frac{1}{2}\sum_{i,j}Count(i,j)-\sum_{i,j}Count(i,j)w_{ij}\\
&\quad +\sum_i\left(\sum_j Count(i,j)\right)\left(\sum_l w_{il}^2\right)\\
&= \frac{N}{2}-\sum_{i,j}Count(i,j)w_{ij}+\frac{1}{2}\sum_{i,j}Count(i)w_{ij}^2\\
&= \frac{N}{2}+\frac{1}{2}\sum_{i,j}Count(i)\left(w_{ij}^2-2\frac{Count(i,j)}{Count(i)}w_{ij}\right)\\
&= \frac{N}{2}+\frac{1}{2}\sum_{i,j}Count(i)\left(\left(w_{ij}-\frac{Count(i,j)}{Count(i)}\right)^2-\left(\frac{Count(i,j)}{Count(i)}\right)^2\right)
\end{aligned}$$

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 $f_i = Count(i)/N$, $f_{ij} = Count(i, j)/Count(i)$

So we have

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

According to the gradient descent rule, we have

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

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

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

The solution to the above differential equation is

$$w_{ij}(t) = f_{ij} + (w_{ij}(0) - f_{ij})e^{-Count(i)t}$$

The above equation can be rewritten as

$$w_{ij}(t) = (1 - e^{-Count(i)t})f + e^{-Count(i)t}w_{ij}(0)$$

References

[1] S. Chen and J. Goodman, "*An Empirical Study of Smoothing Techniques for Language Modeling*" Proceedings of the 34th Annual Meeting of ACL, June 1996

[2] S. Chen and J. Goodman "*An empirical study of smoothing techniques for language modeling*" Technical report TR-10-98, Harvard University, August 1998

[3] J. S. Bridle, "*Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition*". In F.Fogleman Soulie and J.Herault (eds.), Neurocomputing: Algorithms, Architectures and Applications (1990a), Berlin: Springer-Verlag, pp. 227-236.

[4] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", Journal of the Royal Statistical Society B, 39:1-38, 1977

[5] F. Jelinek "*Statistical Methods for Speech Recognition*" Mit Press (1994) pp. 62-70.

[6] R. Kneser and H. Ney "*Improved backing-off for m-gram language modeling*" In Proc. of ICASSP-95, vol. 1, pp. 181-184

[7] S. Lawrence, C.L. Giles, S. Fong, "*Can Recurrent Neural Networks Learn Natural Language Grammars?*" in Proc. of International Conference on Neural Networks, ICNN 96, pp. 1853-1858, June, 1996

[8] A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, A. Oh, "*Creating Natural Dialogs in the Carnegie Mellon Communicator System*". Proceedings of Eurospeech-99, Budapest, Hungary, 1999