Building Language Model Using Error-Correcting Output Codes

Wei Xu, Yi Zhang
School of Computer Science
Carnegie Mellon University
{xw,yiz}@cs.cmu.edu

1. Introduction

Conventional language modeling is based on counting, i.e. estimating the basic n-gram probabilities by formula:

$$1 \le k \le n, \Pr(w_k \mid w_1^{k-1}) = \frac{Count(w_1, w_1^{k-1})}{Count(w_1^{k-1})}. \quad (1)$$

Unfortunately, it is a very inadequate formula, since many possible word n-grams are never actually encountered, especially in a small training dataset. The maximum likelihood estimate for the probability is biased low for unobserved words .If the training dataset is small, we can expect an unsatisfying result.

In this project, we try a different way to build statistical language model. The basic ideas are the following:

1. Build a set of code for each word.
2. Predict the value of each bit of a word given its history in the text.
3. Calculate the conditional probability P(w | h) from the probability of each bit being 0 or 1.

There are several motivations for this project

1. Because the length of code is much less than the number of words, by casting the problem of predicting word to predicting each bit, we can reduce the size of the problem dramatically. Accordingly, the amount of data needed for training can be reduced.
2. By using error correcting output coding, it is possible that we can get better generalization performance on test data.
3. The distributed representation of word is more biologically plausible.
4. It is possible to encode the semantic and syntactic function of word directly into the codes.

This report is organized as follows: In section 2 we introduce a data driven approach to find the word coding. Section 3 describes how a neural network was trained to predict each bit of a word. In section 4 we give a method to calculated the probability of word from the probability of each bit. Then we report our preliminary result in section 5

2. Encoding words

When encoding the words, there are several considerations. First, the Hamming distance of different code should be as big as possible, so that any prediction error for an individual bit can be recovered. Second, We expect that a random coding will make it very difficult to predict each bit. So we want to encode the words in such a way that similar words have similar codes. Here similar words are the words appearing similar context in the data.

2.1. Encoding a single bit

Encoding a single bit is equivalent to partition the vocabulary into two classes. We want to find a partition such that the log likelihood of predicting a word given its previous word

$$LL(\Pi) = \sum_{w_0, w_{-l}} f(w_0, w_{-l}) \log P(c(w_0) \mid w_{-l}) P(w_0 \mid c(w_0))$$

Where

w_{-l} is the lth word in the left context of w_0

$f(w_0, w_{-l})$ is the relative co-occurrence frequency

$P(c(w_0) \mid w_{-l})$ is the probability of a class given its previous word

$P(w_0 \mid c(w_0))$ is the probability of a word given its class

2.2 Encoding multiple bits

When encoding the ith bit, we want the partition introduced by this bit is as much as different to the partition introduced by previous bits.

To define the similarity between two partitions, we need the notion of self-information of a partition and the mutual information between two partitions.

The mutual information of two partitions Π and Π' is

$$MI(\Pi, \Pi') = \sum_{\pi, \pi'} P(w \in \pi) P(w \in \pi' \mid w \in \pi) \log \frac{P(w \in \pi' \mid w \in \pi)}{P(w \in \pi')}$$

Where

π and π' is a class in partition Π and Π' respectively

$P(w \in \pi)$ is the probability that a random word belongs to π

$P(w \in \pi' \mid w \in \pi)$ is the probability that a random word belongs to π' given that it belongs to π

The self-information of a partition Π is

$$H(\Pi) = -\sum_{\pi} P(w \in \pi) \log P(w \in \pi)$$

We define the similarity of two partitions as

$$Similarity(\Pi, \Pi') = 2 \frac{MI(\Pi, \Pi')}{H(\Pi) + H(\Pi')}$$

This similarity measure is symmetric and valued in $[0,1]$

Finally, the quality of a new partition Π can be defined as

$$Quality(\Pi) = LL(\Pi) - \max_i (Similarity(\Pi, \Pi_i))$$

Where Π_i are the partitions we have got so far.

So we want to find a new partition that maximize the Quality. We do the by performing a greedy search followed by simulated annealing.

Note that we are using the lth word w_{-l} in the left context of w_0 when calculating the log likelihood. For different l, we can build different codes. And we concatenate them into one long code.

3. Predicting bit

We use neural network to predict the bits of a word given its history.

The number of input and output units is the length of the code. Each unit in the input and output layer is used to represent one bit of code. Because of the stochastic nature of our problem, the neural net should output the probability of a bit being 1 or 0 instead of giving a hard decision. In each time step, the code of current word is given to the input layer and the output layer gives the probability of each bit of the next word being 1 or 0.

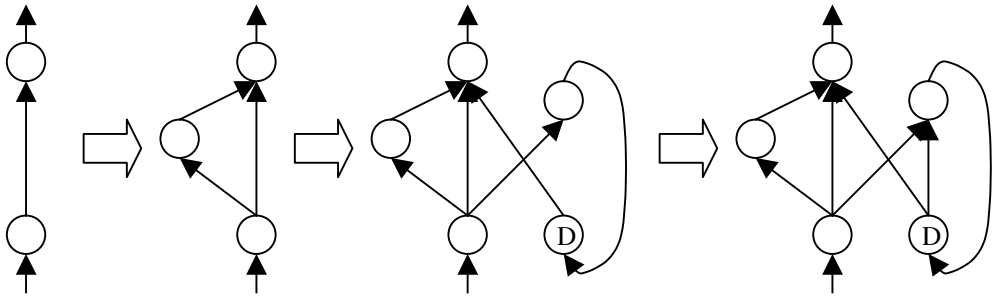
We use cross entropy as cost function instead of square error. We train the network to minimize the cross entropy. The cross entropy is

$$E = \sum_{\mathbf{h}} \sum_{i} \tilde{P}(b_i = 1 \mid \mathbf{h}) \log \frac{1}{o_i(\mathbf{h})} + \tilde{P}(b_i = 0 \mid \mathbf{h}) \log \frac{1}{1 - o_i(\mathbf{h})}$$

Where

$o_i(\mathbf{h})$ is the value of the ith output unit.

$\tilde{P}(b_i = 1 \mid \mathbf{h})$ and $\tilde{P}(b_i = 1 \mid \mathbf{h})$ are the empirical probabilities of the ith bit of a word being 0 and 1 given the history of the word



We use back propagation algorithm to train the network. During training, we gradually increase the capability of the neural net. The final network is obtained through the following four steps

1. We first train a neural net without hidden layer. In this step, the network will first try to capture the linear relation between input and output as much as possible
2. Then we add a hidden layer. The new hidden layer can capture the non-linearity that can not be model by the single layer network
3. Adding another hidden layer and a time delay layer. The output of the new hidden layer is feed to the delay layer. The delay layer will send its input in current time step to the output layer in the next time step. In this way, the network can capture second order time dependency.
4. Adding connection from delay layer to the second hidden layer. Theoretically, this can capture dependency arbitrary long.

Training in each of the above steps is stopped and moved to the next step when the cost function on holdout data reaches lowest point.

4. Inferring word probability from bit probability

First, suppose we have a coding scheme such that all the code

$$g(w \mid \mathbf{h}) = \log \prod_{i=1}^{n} P(b_i = code_w(i) \mid \mathbf{h})$$

$$p(w \mid \mathbf{h}) = \frac{1}{Z(\mathbf{h})} \exp(\lambda_w g(w \mid \mathbf{h}))$$

We need to find the maximum likelihood estimate of $\lambda(w)$. $\lambda(w)$ can be estimated using Iterative Scaling algorithm. We choose not to use Iterative Scaling, but to use the inefficient gradient ascend algorithm for this problem because we can easily use cross validation to prevent overfitting.

The loglikelihood is

$$\begin{aligned} L(\Lambda) &= \sum_{\mathbf{h},w} \tilde{p}(\mathbf{h},w) \log\left(\frac{1}{Z(\mathbf{h})} \exp(\lambda_w g(w \mid \mathbf{h})) \right) \\ &= \sum_{\mathbf{h},w} \tilde{p}(\mathbf{h},w) \lambda_w g(w \mid \mathbf{h}) - \sum_{\mathbf{h}} \tilde{p}(\mathbf{h}) \sum_{w} \exp(\lambda_w g(w \mid \mathbf{h})) \end{aligned}$$

The derivative of $L(\Lambda)$ with respect to λ_w is

$$\frac{\partial L(\Lambda)}{\partial \lambda_w} = \sum_{\mathbf{h}} \tilde{p}(\mathbf{h},w) g(w \mid \mathbf{h}) - \sum_{\mathbf{h}} \tilde{p}(\mathbf{h}) p_{\Lambda}(w \mid \mathbf{h}) g(w \mid \mathbf{h})$$

We update $\lambda(w)$ by

$$\lambda(w) = \lambda(w) + \eta \frac{\partial L(\Lambda)}{\partial \lambda_w}$$

5. Preliminary result

Because all of the three steps are computationally expensive, we choose a small corpus to do the preliminary experiment. The corpus we use is the data collected using the Communicator Telephone Air Travel Information System. 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.

5.1 The coding

We build the codes using the right context and left context with 3 words, i.e. varying l from –3 to 3. The total length of the code is 40 bits. Our coding method can not guarantee that the minimal Hamming distance between two codes is large. In fact, some words even have same code or very similar code in our experiment. The following are some examples of words that have exactly same of very similar codes.

```
E. T. R.
I. F. N.
TO FROM
BIT REGULAR
OTHER LATEST EARLIEST PREVIOUS [ordinary_number] LAST
BRIGHT RED
[airport_code] [city]
RETURN TRAVEL GO FLY
FLYING GOING
```

Upon close examination of these words, we also found that these words appear in very similar context.

5.2 The training of neural net

Each of the output layer and input layer has 40 units. Each of the two hidden layers has 30 units. We are not able to finish all the steps of the training until we finished the report. Here is our current result of the average cross entropy per bit.

	Train	Holdout
step 1	0.6367	0.6368
step 2	0.6366	0.6367
step 3 (still not finish yet)	0.6212	0.6212
step 4 (not begin yet)		

We see that the introduction of delay layer is useful (compared to the introduction of a hidden layer in the step 2)

5.3 Word perplexity

We use the current probability given by the neural network to calculate the word probability. As described in section 4, we need to train λ's using gradient ascend method. And this training has not finished yet. The current perplexity we get is 45.1.

The unigram perplexity of the holdout data is 61.3 and the bigram perplexity on the holdout data is 11.24.

Our result is still far behind the result of stand techniques.

6. Future work

Our approach is only the first step to find the potential of using word coding for language modeling. Although the program is still running, intermediate results are encouraging. Besides waiting for the running result of step 4 of the Neural Network, we can do some further research such as:

1. Try fast training of neural network
2. Other way of predicting bit, e.g., decision tree
3. More advanced encoding, adding semantic and syntactic information to the code using WordNet database.
4. Other ways of calculating word probability from bit probability

Reference

Miller, George A., Richard Beckwith, Christiane Fellbaum, Derek Gross and Katherine J. Miller. ''Introduction to WordNet: an on-line lexical database." In: *International Journal of Lexicography* 3 (4), 1990, pp. 235 - 244. ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps
Keywords:

Plaut, D. C. and Shallice, T. (1994). Word reading in damaged connectionist networks: Computational and neuropsychological implications. In R. Mammone (Ed.), *Artificial neural networks for speech and vision* (pp. 294-323). London: Chapman and Hall.