# Count-Based and Neural Network Language Models

**Vincent Nguyen** and **Austin Shin**
CS 287: Statistical Natural Language Processing
Harvard University
Cambridge, MA 02138
{vnguyen01, ashin01}@college.harvard.edu

## 1 Introduction

Language models represent the an entire probability distribution over words and are useful for modeling the probability of sequences or next word prediction tasks. Simple language models rely on counts of words while more complex models take into account the distributional context representation of sequences. We explore several different models and show their predictive performances on the experimental data.

## 2 Data

We use the Penn Treebank data for the part of speech tagging task. The Penn Treebank is a large annotated corpus of English text contained nearly 2,500 stories from the Wall Street Journal [1].

We split the data into train, valid, and test sets. The train set consists of sentences from stories that we can use to build our language model for this corpora. For each valid set, we have a series of words leading up to a word prediction. There are 50 possible words to choose from and an index for the correct word. The Kaggle competition relies on a perplexity score for these 50 words, but we use the entire distribution for the perplexity scores mentioned in this assignment.

## 3 Learning Methods

### 3.1 Count-Based Models

Count-based models (CBM) represent the most basic method of language modelling. Such sparse models achieve quick performance on large corpora with surprising predictive performance. CBM are dependent upon counting the number of word occurrences in order to generate probabilistic predictions of future words given some context history.

**N-grams** The term $n$-gram represents a continuous sequence of $n$ word features in a sentence. Since CBM can suffer from failing to capture the distributional context of words, $n$-gram size can in many cases help to represent a shortened context history.

For example, given a corpus with the single sentence

```
the quick brown fox jumps over the
lazy dog
```

a unigram model will split the sentence into single words and their relative frequencies will the the probabilities of them occurring [2]. N-grams of size 2 will have two contiguous words together as one feature. The counts of these two-word features will then be divided by the counts of each single unigram preceding the second word. Formally, we define this as

$$F_{c,w} = \sum_{i-1}^{|V|} \mathbf{1}(w_{i-1:i} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-1:i} = c, \theta) = \frac{F_{c,w}}{F_{c,*}}$$

For trigrams, we would be taking three continguous words together as one feature and dividing by the context of the previous two words so that

$$F_{c,w} = \sum_{i-1}^{|V|} \mathbf{1}(w_{i-2:i-1} = c, w_i = w)$$

$$p_{ML}(w_i | w_{i-2:i-1} = c, \theta) = \frac{F_{c,w}}{F_{c,*}}$$

For the purposes of this assignment, we work with bigrams and trigrams.

**Maximum Likelihood Estimation** The probability of predicting the next word given previous words can be efficiently calculated in CBM with Maximum Likelihood Estimation (MLE). MLE of some future word $w_i$ occurring is simply the probabilities drawn from our feature set and denoted above.

[1] https://catalog.ldc.upenn.edu/LDC99T42

[2] Note that unigrams, $n$-grams of size 1 completely fail to capture any information predicting future words.

$$p_{ML}(w_i) = \frac{c_i}{N}$$

For the case of bigrams, we can construct a sparse square count matrix $\in \mathbb{R}^{|V|\times|V|}$. Each row represents a context word and each column is the count of words that occur after the context word. We normalized this bigram count matrix by the context counts. The next word probability can be calculated by simply looking up by row and column.

For the case of trigrams, we can imagine a tensor $\in \mathbb{R}^{|V|\times|V|\times|V|}$ where we $|V| \times |V|$ rows which represent all bigrams and each column entry is a vector of $|V|$ columns represent all possible third words. Each of these vectors would be divided by the number of bigram counts and the probability can be looked up similar to the previous case. This abstract conception will not instantiate in many cases due to the large feature vocabulary and hashing tricks need to be employed.

**Laplace Smoothing** CBM place a large emphasis on smoothing techniques since there are many cases when the number of counts are 0 which leads to an MLE of 0. Giving a probability of 0 to unseen words arises frequently in sparse models and smoothing can adjust this penalty. We add an $\alpha$ to the number of counts and divide by the total plus the vocabulary size to smooth out unseen words.

$$p_{Laplace}(w_i) = \frac{c_i + \alpha}{N + \alpha|V|}$$

For rare contexts, this runs into the probability of sharply shifting probabilities.

**Witten-Bell** To better deal with unseen words which receive a probability of 0, the Witten-Bell estimation attaches a parameter $\lambda(c,w)$ which must sum to 1 and a backoff probability. We define

$$(1-\lambda) = \frac{N_{c,\cdot}}{N_{c,\cdot} + F_{c,\cdot}}$$

$$p_{wb}(w_i) = \frac{F_{c,w} + N_{c,\cdot} \times p_{wb}(w|c')}{F_{c,\cdot} + N_{c,\cdot}}$$

when we come across unseen words, we are effectively using the backoff distribution probability. The alternative in Laplace smoothing is to shift probability mass from seen events to unseen events.

## 3.2 Neural Network Language Models

**Bengio NNLM** We follow the Neural Network language model as described by Bengio et. al. with a fixed window size, $d_{win} = 5$. The inputs to the model are a window-length sequence of the word-ids. Each id is transformed to a 30-dimensional word-embedding via a lookup table.

$$x_{embed} \in R^{30}$$

The embeddings are then concatenated together, and fed through a linear layer that transforms to a dimension of size $d_{hidden}$ which in our case we chose to be 100, following Bengio's work.

$$y_1 = W_1[x_{embed_1}, ..., x_{embed_5}] + b_1$$

where $W_1 \in R^{150x100}$. This result is passed through a non-linearity (tanh), followed by another linear layer, $W_2 \in R^{100x|V|}$ which maps to the output classes, which in this case is the vocabulary. Finally, we add a softmax to force a distribution.

$$y_{out} = softmax(W_2(tanh(y_1)) + b_2)$$

To train, we use stochastic gradient descent on the cross-entropy criterion to compare the predicted distribution, and the correct prediction one-hot distribution.

**Noise Contrastive Estimation** The NNLM suffers from slow evaluation on both the forward and backward propagation phases of training. This is largely due to the calculation of the partition function during the softmax, which must be computed over all classes (in this case, all words in the dictionary).

To deal with this, we employ a model which instead attempts to detect whether a given context-word pair is generated by the true distribution, or a noise distribution. More formally, consider the dataset: $((x_1,y_1),D_1),...,((x_n,y_n),D_n)$, where $x_i$ is a context, $y_i$ is a candidate word, and D is 1 if the $y_i$ came from the true distribution and 0 otherwise. Then, we wish to minimize the cross-entropy between our predicted D's and the training set,

$$Loss = \sum_i L_{cross-entropy}(D_i, \hat{D}_i)$$

In this case, our model is identical to the one above except that we remove the softmax and work only with the z-scores provided by the second linear layer.

Under certain conditions, the gradients provided by this objective function may still well-approximate the gradients provided by our original objective. In particular, if we sample k noisy words from a uniform distribution over the vocabulary for every context, true-word pair, then our gradients will be:

$$\frac{dL}{d\theta} = \frac{k}{|V|z(w) + k} \frac{d}{d\theta} \log z(w) - \sum_{i=1}^{k} \frac{z(wd_i)}{z(wd_i) + k/|V|} \frac{d}{d\theta} \log z(wd_i) \quad (1)$$

where $z(w)$ is the unnormalized z-score of the true word and $z(wd_i)$ is the unnormalized z-score of the i'th noisy word. To see how these gradients are related, we find:

$$\lim_{k\to\infty} \frac{dL}{d\theta} = \frac{d}{d\theta} \log z(w)$$

which is the gradient of our original NNLM loss function.

# 4 Results

## 4.1 Perplexity

Rather than using raw accuracy, we evaluate the models using perplexity,

$$perp = \exp(-\frac{1}{n}\sum_{1}^{n}\log p(w_i|w_1, w_2, ...w_{i-1})$$

a metric that is the average negative log-likelihood. Our language models predict a distribution over the possible next words, and minimizing perplexity is the same as maximizing probability. The log base 2 of perplexity is the cross-entropy in bits per word.

## 4.2 CBM

**Simple MLE** CBM only achieve high performance when the test or validation set is very similar to the train set. For bigrams, a simple MLE probability from counts divided by the total does not work without a slight smoothing parameter. This is due to the fact that there were many unseens words. Therefore, we would be dividing by 0 when normalizing the bigram counts by its context. We add the value of 1 to all counts to arrive at a "simple" MLE estimate.

| CBM | Smoothing | Valid Perplexity |
|---|---|---|
| Simple MLE Bigrams | None | nan |
| "Simple" MLE Bigrms | 1 | 250.0142 |
| "Simple" MLE Trigrams | 1 | 1047.0231 |

Note that this pseudo-smoothing is different from Laplace smoothing.

**Laplace Estimate** We estimate using Laplace smoothing for different $\alpha$ parameters. As we go

| CBM | Smoothing | Valid Perplexity |
|---|---|---|
| Bigrams | 1 | 242.8090 |
| Bigrams | 2 | 236.0846 |
| Bigrams | 5 | 233.1046 |
| Trigrams | 1 | 942.2012 |
| Trigrams | 2 | 970.9381 |
| Trigrams | 5 | 1045.2841 |

from bigrams to trigrams, smoothing becomes more and more important due to the frequency on unseen sequences. However, we notice that increasing $\alpha$ for trigrams results in much worse perplexity scores. A possible explanation for this is that probability mass is reserved for unseen events. For the case of trigrams too much probability mass is taken from seen events and given to unseen events.

| CBM | Valid Perplexity |
|---|---|
| WB-Bigrams | 407.2901 |
| WB-Trigrams | 570.1310 |

**Witten-Bell Estimate** We were surprised to see Witten-Bell perform worse than Laplace smoothing. We hypothesize that this may be due to possible errors in calculation for unseen events. Another possible explanation may be that there is still too much probability mass being shifted to unseen events. A possible exploration would be to investigate low-smoothing techniques such as discounting.

## 4.3 NNLM

To speed up performance of our NNLM, we use Adam instead of SGD. This results in significant performance speeds, especially when training in batches greater than 1000. Note that in the table, the 30 in NNLM-30 refers to the embedded dimension size.

| NNLM | $d_{hid}$ | Valid Perplexity |
|---|---|---|
| NNLM-30 | 100 | 181.27 |
| NNLM-100 | 100 | 180.78 |

## 4.4 NCE

While we were able to construct the NCE model, we were unable to train the NCE. After much struggle, we believe our failure is due to poor supplied gradients rather than a misspecified model since not only does the NCE loss function increase, the cross-entropy loss for the original model also increases over time. We attempted to train the model using various optimization techniques and hyperparameter settings, suggesting that the gradients were poor. Some of this error was mitigated by the use of the propagation built into Torch, however the input gradient to the model required for the custom loss function had to be done by hand.

## 4.5 Word Babblers

As a fun extension, we try word babblers for bigrams and trigrams. Starting with the word `now`, we ran into issues where the babbling would always run into the 2 and 4 tags, `</s>` and `</unk>` respectively. When using the NNLM without NCE, we also ran into the same issue and did not come up with very meaningful sentences. The only way to arrive at long sentences was to start with rare words.

## 4.6 Kaggle

As part of the in-class Kaggle competition, we also submitted various models. As of submission, we are currently 11th on the leaderboard (50% of the test data). We

do not show all our submissions because several were done to test if our perplexity metric evaluated correctly. Several of the top scores use NCE for the NNLM which

| Model | Scores |
|---|---|
| NNLM | 2.45006 |
| NNLM | 1.73999 |
| NNLM | 1.56883 |
| NNLM | 1.53188 |
| NNLM | **1.35977** |
| Bigrams | 1.92376 |
| Bigrams | 2.24949 |

we were unable to implement.

## 5 Conclusions

The NN model as presented by Bengio et al. is an effective lanugage model for the English language as we've seen. In this paper we have also explored the effectiveness of this model relative to count-based n-gram models that do not take into account higher-order interactions between words. These models demonstrated much weaker performance, although they approached the performance of the NN model when smoothing was implemented.

Some of the limitations of the NN model were its speed. Explicit calculation of the partition function is expensive, and so Noise Contrastive Estimation was employed to approximate the gradients for the parameters. Our attempt was ultimately unsuccessful, and is a source of grave disappointment. We will attempt to fix this.

In the future, we would like to explore the effectiveness of pretraining the embeddings, as those represented a large portion of the total number of parameters in the model.