

HW3: Language Models

Sebastian Gehrmann
gehrmann@seas.harvard.edu

March 7, 2016

1 Introduction

A crucial task in natural language processing is language modeling. Language modeling means calculating the probability of a sentence. This is important for many problems in speech recognition, machine translation, text summarization and many more.

A standard approach to this problem is calculating the probability of a word given the previous n words. This reduces the problem to a classification problem of a special kind. It is special in the sense that it has many possible classes (all words in a vocabulary) in which there is not a correct class. The goal is to model the correct probability distribution over the next word (class).

In this paper, we present and compare two different approaches. The first is building a count-based language model with different smoothing methods to attack the sparsity of the data. The second is the neural language model - a neural network that takes as input the previous n words and produces the probability distribution over the following word (Bengio et al., 2003).

The code can be found at <https://github.com/sebastianGehrmann/CS287-HW3> and the capability of it can be seen in the kaggle competition with the username "Sebastian" (the current best model is a neural network with trigrams).

2 Problem Description

A sentence is composed of n words, all of which can syntactically and semantically influence each other. There are even longer dependencies over more sentences or even paragraphs. Thus, in order to have a perfect model of a language, one has to take into account all words. To do this, the probability of a sentence is defined as the product of all the probabilities of its words.

That means, one has to compute $p(w_i|w_1, \dots, w_{i-1})$ - that is the probability of a word given all the previous words in a text. This leads to a problem of sparsity. What is the probability of a word if we have not seen it in this exact context?

To approach this problem, we hypothesize that the words in closer proximity have most of the influence on the probability. We can therefore disregard most of the context and not many information. Taking only n words into account is called n -gram model which is defined as

$$p(w_i|w_{i-n+1}, \dots, w_{i-1})$$

This yields a probability distribution over the whole vocabulary of words for a given task. Since there is not one perfect word that can come next, but many words with different probabilities, the prediction accuracy is no longer a valid method of evaluating the performance of the model. Instead, we compute the perplexity. Perplexity is defined as e to the power of the average negative log-likelihood (NLL) $perp = \exp(-\frac{1}{n} \sum_{i=1}^n \log p(w_i|w_1, \dots, w_{i-1}))$.

3 Model and Algorithms

The first approach we report on is a multinomial estimate of $p(w_i|w_{i-n+1}, \dots, w_{i-1})$. Since the data is sparse, we need to deal with previously unseen contexts. This is done by using different smoothing methods. In this paper we compare the three methods

- Simple maximum Likelihood estimation
- Estimation with Laplace Smoothing (α)
- Estimation with Witten-Bell

The second approach is modeled after the work of Bengio et al. (2003) which is a neural network that takes as input the contexts and the words to learn the probability distribution over the next word.

3.1 Maximum Likelihood

This is the most simple method of estimating the probabilities for a word. The maximum likelihood estimation is completely count based. Such, we count the occurrences of every context F_c , and the occurrences of every context with a word $F_{c,w}$.

To calculate $p_{ML}(w|c)$, we only have to calculate $\frac{F_{c,w}}{F_c}$. Intuitively, we just learn the ratio of how often that word occurs given the context and compare it to how often we see the context in general.

While this makes a lot of sense, we don't attack the problem that this estimate is always only as good as our training data. Since the number of possible combinations of words is growing exponentially with the size of the n-gram, the data will be incredibly sparse and many of the probabilities will be 0. This makes it necessary to smooth the counts.

3.2 Laplace Smoothing

Laplace smoothing is the easiest way to attack the problem of unseen words. We assume that we have seen every word-context combination exactly α times more than we have actually seen it, and increase the count for the contexts subsequently by $\alpha|\mathcal{V}|$ where \mathcal{V} is the vocabulary.

However, this punishes frequent words and more than proportionally benefits rare words as all words are treated equally. To get around this, we use a third method called Witten-Bell.

3.3 Witten-Bell

The idea of smoothing methods like Witten-Bell is to use the backoff of a context (context without first word) and interpolate the maximum likelihood probabilities using different backoffs. That means that we combine for a trigram the trigram, bigram and unigram probabilities and multiply

them by a parameter so that the result is still a probability distribution. To put it in mathematical form, the interpolation probability is

$$p_{interp} = \lambda_1 p_{ML}(w|c) + \lambda_2 p_{ML}(w|c') + \lambda_3 p_{ML}(w|c'')$$

To ensure the convexity of the combination we set the lambdas so that

$$\sum_i \lambda_i = 1$$

$$\lambda_i \geq 0 \forall i$$

In the case of Witten-Bell, we use a special form for λ by setting

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

Here, N is the count of unique words seen in a context which is different from F which is the total number of times we have seen the context. By inserting this into the interpolation formula, we can define the probability recursively and n-gram size independently as

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

Now, we have an estimation for new words that is proportional to the probability of seeing a new word. That way, common contexts punish rare words while rare contexts are more liberal in giving a higher probability during the smoothing.

3.4 Neural Language Model

N-gram models suffer from the common problem in NLP that shared features between words are not exploited. As an example, the sentences “The dog sleeps” and “A cat naps” are completely different for the count-based models even though they are semantically very similar. Not even backoff would help because they do not share any words. We can get around this flaw by using word-embeddings and a neural network to learn the interactions between the words. Using the words $f_1, \dots, f_{d_{win}}$ in the window, we concatenate their embeddings to have position-sensitive weights to learn. This concatenation is \mathbf{x} . The network is a standard multilayer perceptron, which means that the output looks like

$$NN_{MLP} = \tanh(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)\mathbf{W}^2 + \mathbf{b}^2$$

4 Experiments

Finally we end with the experimental section. Each assignment will make clear the main experiments and baselines that you should run. For these experiments you should present a main results table. Here we give a sample Table 1. In addition to these results you should describe in words what the table shows and the relative performance of the models.

Besides the main results we will also ask you to present other results comparing particular aspects of the models. For instance, for word embedding experiments, we may ask you to show a chart of the projected word vectors. This experiment will lead to something like Figure ?? . This should also be described within the body of the text itself.

Model	Acc.
BASELINE 1	0.45
BASELINE 2	2.59
MODEL 1	10.59
MODEL 2	13.42
MODEL 3	7.49

Table 1: Table with the main results.

5 Conclusion

End the write-up with a very short recap of the main experiments and the main results. Describe any challenges you may have faced, and what could have been improved in the model.

References

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.