HW3: Language Models

Sebastian Gehrmann
gehrmann@seas.harvard.edu

March 9, 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). Different predictions can also be seen on the cluster in the folder `/n/regal/rush_lab/sebastian`. The names of the files are of the form EPOCH-DWIN-PERP-LOSS.h5.

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

3 Model and Algorithms

The first approach we report on is a multinomial estimate of $p(w_i|w_{i-n+1}, \ldots, w_{i-1})$. 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. We also try to implement noise-contrastive estimation (Gutmann and Hyvärinen, 2010), which similar to word2vec learns to distinguish between correct and incorrect samples.

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 $F_{c,\cdot}$ and the occurrences of every context with a word $F_{c,w}$.

To calculate $p_{ML}(w|c)$, we only have to calculate $\frac{F_{c,w}}{F_{c,\cdot}}$. 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 is 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. Be 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. The model in this paper is the same as in Bengio et al. (2003) 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 (MLP), which means that the output looks like

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

To force the distribution, we define $\hat{y}$ as the softmax of the result of the MLP. The model is trained by minimizing the Cross-entropy using minibatch-SGD

3.5 Noise-Contrastive Estimation

A downside of the neural language model is the long time it needs to train. This is because for every prediction, it has to compute the softmax over all potential output classes (in the partition function) to force a distribution. The idea behind noise-contrastive estimation (NCE) is to use the scores $\mathbf{z} = \tanh(\mathbf{x}\mathbf{W}^{\mathbf{1}} + \mathbf{b}^{\mathbf{1}})\mathbf{W}^{\mathbf{2}} + \mathbf{b}^{\mathbf{2}}$ directly. (Gutmann and Hyvärinen, 2010)

The goal of the scores is to reinforce correct predictions and punish wrong predictions just as in ranking-loss. We can do this by using the sigmoid-function σ on the scores $\mathbf{z}$. Thus, we have to for every test-batch create a data set of the form $((\mathbf{x_1}, \mathbf{y_1}), \mathbf{d_1}), \ldots, ((\mathbf{x_n}, \mathbf{y_n}), \mathbf{d_n})$ in which y is the class for which we want to calculate the score and d says whether the y was the correct class. Predicting all these scores will yield a prediction $\hat{d}$, which we then can compare to the actual d using cross-entropy loss. Implementation-wise, this is similar to a neural language model. However, instead of computing the full $\mathbf{W}^{\mathbf{2}}$ plus softmax, we only use a lookup for the samples in the current dataset $\mathbf{W}^{\mathbf{2}}_{:,\mathbf{w}}$ and compute σ.

The probability of a true sample can be split as

$$p(d = 1|\mathbf{x}, \mathbf{y}) = \sigma(\log p(\mathbf{y}|d = 1, \mathbf{x}) + \sigma(\log p(\mathbf{y}|d = 0, \mathbf{x})$$

To generate a probabilistic sample, we introduce a noise variable K that defines the prior probability of a sample being correct or incorrect. That is, $p(correct) = \frac{1}{1+K}$ or K wrong samples per correct sample. Mnih and Teh (2012) show, that the loss for this model can be described as

$$\mathcal{L}(\theta) = \sum_i \log \sigma(z_{w_i} - \log(K_{p_{ML}}(w_i)) + \sum_{k=1}^{K} \log(1 - \sigma(K_{p_{ML}}(s_{i,k}))$$

This still leaves some questions regarding an efficient implementation. Luckily, both z_w and $\log p_{ML}(w)$ can be looked up in a table. Since the scores are now no longer normalized, in order to compute the whole distribution, we can use the softmax again or compute the scores for a subset of the data which we will do in the experiments.

4 Experiments

We first report the results of the count-based models. Since the models are deterministic, there are not many parameters to try out. We report the perplexity on n-grams of sizes 2 and 3 on a smaller set of possible output classes (50).

The results on the validation set can be seen in table 1. The baseline was chosen as the perplexity of a unigram prediction which is equal to the number of the possible classes. The lower the perplexity the better.

Model	2-Gram	3-Gram
Baseline	50.00	50.00
MLE	9.42	16.63
Laplace $\alpha = 0.1$	13.24	24.50
Laplace $\alpha = 1$	20.88	35.32
Witten-Bell	16.41	17.36

Table 1: The results for count-based methods

The second experiment was running the neural language model. We trained it with 2-grams, 3-grams and 5-grams. Additionally, two different typed of regularization were used. The first was no regularization, the second one was the renormalization of the l2-norm of the word embeddings after each epoch. The results of this experiment are shown in table 2.

Model	2-Gram	3-Gram	5-Gram
Baseline	10,000	10,000	10,000
NNLM	251.1	253.1	250.27
NNLM regularized	190.3	189.4	191.4

Table 2: The perplexity for the regularized and ne non-regularized NNLM

The perplexity of the regularized model with the smaller set is around 1, which means that is is extremely certain of the next coming word, regardless of the size of the n-gram.

5 Conclusion

As we can see in the count-based methods, a simple maximum-likelihood estimation is often a very good estimation, even outperforming methods like laplace-smoothing. Estimating the correct smoothing is a hard process and it assumes many things that are not necessarily true. However, since all count-based approaches are deterministic, the training is very fast, way faster than neural network approaches.

The neural network approaches however, outperform the count-based models and are very easy to scale to larger n-grams (even though in this task it does not seem to improve the perplexity too much).

References

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

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *International Conference on Artificial Intelligence and Statistics*, pages 297–304.

Mnih, A. and Teh, Y. W. (2012). A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.