Neural Probabilistic Language Models for Speech Recognition
DT2119 Project Report

Dmytro Kalpakchi
dmytroka@kth.se

Abstract

Language model is an important component of any speech recognizer, as it navigates the search among alternative word hypotheses. A traditional approach relies on n-grams that use previous $n-1$ words to get a probability distribution for an n^{th} word using maximum likelihood estimate. N-grams are still very popular because they're computationally cheap and lead to successful results. However, as the order of n-grams increases, the problem of data sparsity becomes critical. In order to generalize to unseen cases different kinds of smoothing, back-off and interpolation methods are used. In this paper a different approach to addressing these unseen cases is considered, based on neural probabilistic language models.

1 Introduction

If a person says

> Could you, please, close the ...

you would probably expect to hear *door* as the next word more than *floor*. However it's harder to differentiate such cases for speech recognizer given that a-priori probabilities for all words are equal. Therefore, language model (LM) that assign meaningful prior probabilities to each possible next word, given a text, is of major interest. The class of LMs that deals with sequences of words in a probabilistic way is called *stochastic language models*. The most popular model in this class is n-gram LM which estimates the probability of the next word given $n-1$ previous using maximum likelihood estimate (MLE):

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\text{count}(w_{i-n+1}^{i})}{\text{count}(w_{i-n+1}^{i-1})} \tag{1}$$

Researchers usually get state-of-the-art results by setting $n = 3$, i.e. using trigrams. However, pure 3-grams (and n-grams in general) can't deal with new combinations of words, unseen in training data, as an MLE of their probability is zero. This is the effect of data sparsity, as the number of these unseen (or out-of-vocabulary, OOV) cases grows with the order of n-grams. Smoothing is a traditional way to deal with this problem. Smoothing techniques are smart tricks aiming to redistribute probability mass from seen to unseen words.

It is argued in [7] that there are at least two properties of such approach that "beg to be improved upon":

1. wider contexts should be used (2 words are definitely not enough);
2. semantical and/or grammatical similarity between words should be taken into account.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

The idea of the last point is that from *"Mom is going to cafe"* we should be able to generalize to *"Mom was going to restaurant"*, as *"cafe"* and *"restaurant"* are semantically similar, *"is"* and *"was"* have similar grammatical roles.

The idea of exploring semantical and grammatical similarities of words is very popular in modern research. Semantical and grammatical structure of the language is explored by embedding each word in a high-dimensional space where each dimension is responsible for several semantical and/or grammatical aspects. Two very famous models are word2vec introduced in [10] and GloVe introduced in [11]. One of the papers that laid foundation for this "word vector revolution" is a paper [7], used as a base for this project.

Briefly, in [7] it is proposed to learn both word vectors (in high-dimensional space) and probabilities for all words in the same time. The model is supposed to generalize by assigning high probability to an unseen sequence if it consists of similar words. The models for achieving two described goals are called *neural probabilistic language models* (NPLM). NPLMs were expected to perform well for speech recognition [7], and later in [12] and [8] they were used for speech recognition and outperformed n-grams.

In this project we explored NPLM as a language model for speech recognition and compared it to trigram language model smoothed using Chen and Goodman's modified Kneser-Ney discounting [9]. Both LMs were evaluated outside of speech recognition domain using perplexity metric (intrinsic evaluation) and in the context of speech recognition using word error rate (extrinsic evaluation).

2 Neural Probabilistic Language Model

Formally, NPLM is defined in [7] as follows. Let V be the vocabulary of all words $w_i \in V$ in the training set $w_1, \dots, w_N$. We want to learn a statistical language model that computes the probability of the next word given $n-1$ previous and produces high likelihood for OOV cases. Considering Markov assumption, the model can be formally written $f(w_i, w_{i-n+1}^{i-1}) = P(w_i | w_{i-n+1}^{i-1}) = P(w_i | w_1^{i-1})$. The model f has only two constraints for any w_1^{i-1}:

$$\sum_{j=1}^{|V|} f(w_i = j, w_{i-n+1}^{i-1}) = 1 \tag{2}$$

$$f(\cdot, w_{i-n+1}^{i-1}) > 0 \tag{3}$$

Let's decompose the function $f(w_i, w_{i-n+1}^{i-1})$ into two parts:

1. a mapping LT from any word $w_i \in V$ to a *distributed feature vector* $LT(w_i) \in \mathbb{R}^m$;
2. the probability function g that maps a sequence of distributed feature vectors for words in context $(LT(w_{i-n+1}), \dots, LT(w_{i-1})$ to a conditional probability distribution for the next word w_i over all possible words in V, i.e.:

$$f(w_i = j, w_{i-n+1}^{i-1}) = g(w_i = j, LT(w_{i-n+1}^{i-1})) \tag{4}$$

This composition of two functions g and LT can be represented with a neural network with the architecture shown on the figure 1.

Each of the layers has parameters Θ associated with it. Parameters of the *embedding layer*, corresponding to the mapping LT, are feature vectors for the word in context, usually represented by a $|V| \times m$ matrix T where row T_i represents a feature vector $LT(w_i)$. Interestingly a function g can be a neural network itself with feed-forward or recurrent topology and generic parameters ξ. The network is trained by maximizing the regularized log-likelihood L w.r.t. Θ:

$$L = \frac{1}{N} \sum_i \log f(w_i, w_{i-n+1}^{i-1}; \Theta) + R(\Theta), \tag{5}$$

where $\Theta = \{T, \xi\}$ and $R(\Theta)$ is a regularization term.

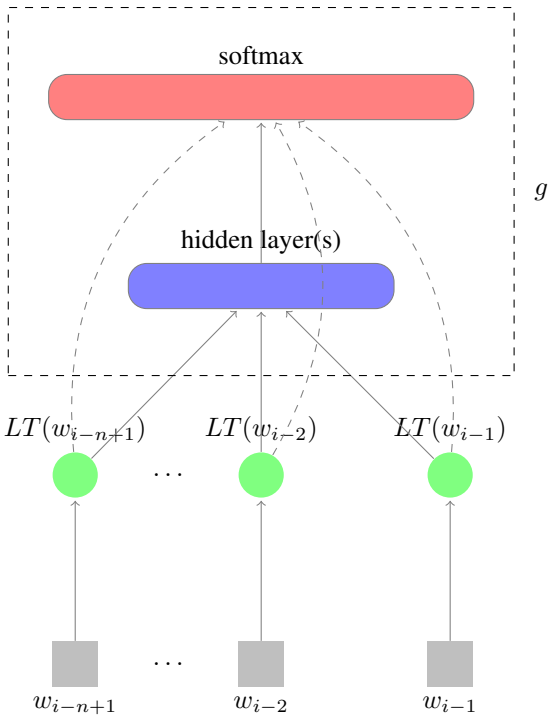

Figure 1: Architecture of neural network $g(w_i = j, LT(w_{i-n+1}^{i-1}))$

The number of hidden layers in feed-forward architecture may vary. Another possibility is to implement function g as a recurrent neural network. For the purpose of this project, it was chosen to use the architecture with two hidden layers. The decision was inspired by its successful application in machine translation [13] (outperformed a baseline 5-gram model) and speech recognition [12] (performed better than 3-grams, but worse than 4-grams). It leads to a thought that two hidden layer networks might capture grammatical and semantical similarities better leading to more meaningful probabilities of final language model.

3 Experiments

It was decided to focus only on training NPLMs based on trigrams, i.e. feeding two words as input and trying to predict the third one. In all experiments neural network for NPLM training had 2 hidden layers, both with *tanh* activation function. Input words were embedded into 30-dimensional space, the first hidden layer contained 100 neurons, the second - 30 neurons. NPLMs were trained for 15 epochs on datasets train2580 and train25800 and only for 8 epochs on dataset train258000, because of computational reasons (took 20 hours to train 8 epochs on PDC).

All trained NPLMs were compared with baseline trigram back-off language model that used modified Kneser-Ney discounting [9] as a smoothing technique.

3.1 Data

For language models training and testing, subsets of Google's 1 Billion Word Language Model benchmark dataset [1] were used. Subsets of larger size include the sentences from the subsets of all smaller sizes. The characteristics of all training subsets are described in table 1. In order to give a language model a chance to learn some probability of unseen words, the vocabulary of k most frequent words was chosen for training subsets where k is the closest divisible by 500 number to the number of unique words in training set, but no more than 60000 words.

Table 1: 1 Billion Word Language Model benchmark dataset training subsets' characteristics

ID	Number of sentences	Number of unique words	Vocabulary size
train2580	2580	9684	9500
train25800	25800	34266	34000
train258000	258000	102843	60000
test	287562	107980	-

The same vocabulary is then used to train a trigram model with only difference that 3 technical words are added for NPLM:

- <s> to denote the start of a sentence;
- </s> to denote the end of a sentence;
- <unk> to denote OOV cases.

For the extrinsic evaluation of language models (in the context of speech recognition) the VoxForge open source dataset [2] was chosen. Language model was evaluated on the test set of 8kHz wav audio files using acoustic models (version 0.1.3) for CMU Sphinx trained by VoxForge. The train/test split defined by VoxForge was used to maintain the validity of performed evaluation.

Table 2: VoxForge dataset characteristics

Number of training audio files	42142
Number of test audio files	4682

3.2 Intrinsic Evaluation

Perplexity (PP), used for an intrinsic evaluation of the LM, is defined as,

$$PP(w_1^N) = P(w_1^N)^{-1/N} = \prod_{i_1}^{N} P(w_i|w_1^{i-1})^{-1/N} \tag{6}$$

where N is the number of words in the test data for calculating perplexity.

Due to the inverse in eq. 6, higher likelihood of a word sequence implies lower the perplexity. Therefore, maximizing the likelihood, which is a part of NPLM's objective function (5), leads to minimizing a perplexity of LM.

3.3 Extrinsic Evaluation

However, improvement of perplexity doesn't imply a proportional improvement of speech recognition accuracy. Therefore, evaluation in context of speech recognizer is needed. Word error rate (WER) is usually used for such evaluation and is defined as follows.

$$\text{WER} = \frac{S + I + D}{N} \tag{7}$$

where

- S is the number of substitutions,
- I is the number of insertions,
- D is the number of deletions, and
- N is the total number of words.

However, it's impossible to evaluate WER of only trigrams using CMU Sphinx, that's why, in order to evaluate NPLMs extrinsically, a rescoring procedure was used. Rescoring for evaluating NPLMs in speech recognition was described in detail in [8]. The output of NPLM was limited to k most frequent

unique words in the training set, called a shortlist. For all the words in a shortlist the probabilities $P_{\mathrm{NPLM}}(w_t|h_t)$ were calculated by NPLM for the case of $|h_t| = 2$, i.e. trigrams. To get a rescored model, a baseline trigram model was used with probabilities $P_B(w_t|h_t)$ for each uni-, bi- and trigram. Uni- and bigrams were transferred to a rescored model as-is. For trigrams the probability mass of all words in a shortlist was redistributed according to the equation below.

$$\hat{P}(w_t|h_t) = \begin{cases} P_{NPLM}(w_t|h_t) \cdot P_S(h_t), \text{ if } w_t \text{ is in shortlist} \\ P_B(w_t|h_t), \text{ otherwise} \end{cases} \tag{8}$$

where $P_S(h_t) = \sum_{w \in shortlist(h_t)} P_B(w|h_t)$ and $|h_t| = 2$, as we're focused on trigrams. As all the probabilites in ARPA language model format are calculated in log-domain, the (8) changes to:

$$\log \hat{P}(w_t|h_t) = \begin{cases} \log P_{NPLM}(w_t|h_t) + \log P_S(h_t), \text{ if } w_t \text{ is in shortlist} \\ \log P_B(w_t|h_t), \text{ otherwise} \end{cases} \tag{9}$$

where $\log P_S(h_t) = \log(\sum_{w \in shortlist(h_t)} P_B(w|h_t))$

3.4 Technologies used

NPLM toolkit [4] was used for training NPLMs. Trigram back-off models with Kneser-Ney smoothing and perplexity of all LMs were evaluated using SRILM toolkit [6]. WER was evaluated using pocketsphinx, part of CMUSphinx, open source speech recognition toolkit [5]. All language models were stored in ARPA format [3].

4 Results

Results of intrinsic and extrinsic evaluation of NPLMs vs baseline model are presented in table 3.

Table 3: Perplexity and word error rate of NPLMs trained on datasets of different sizes.

Model	Dataset	Test set PP (pure)	Test set PP (rescored)	WER
3-gram back-off (baseline model)	train2580	515.097	-	55.89%
	train25800	532.7537	-	46.10%
	train258000	349.3361	-	40.94%
NPLM	train2580	667.282	636.4265	57.08%
	train25000	1041.97	958.7714	49.01%
	train250000	1083.59	1122.079	46.10%

Evidently, models, re-scored with NPLMs perform worse than the baseline model. NPLMs have higher perplexity on the test set and higher WERs on VoxForge test dataset. However, test perplexity of rescored models does not give a meaningful proxy to a word error rate (the same anomaly is reported in [12]). The reason for this is that the number of trigrams is quite small compared to uni- and bigrams which means that WER should not be influenced that much, while perplexity, being probabilistic metric, may suffer after re-scoring. In order of investigate if bad performance can be explained by insufficient amount of training epochs, the log-perplexity over training epochs was plotted in figure 2.

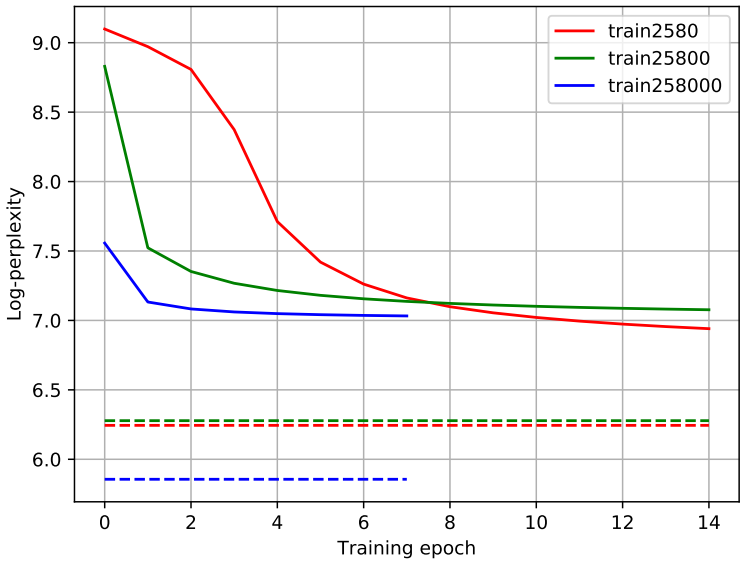

Figure 2: Perplexity (in log-domain) on training set over training epochs for NPLM. Dashed lines represent the corresponding baseline model perplexity

As one can see, perplexity does not fluctuate during training and has a clear diminishing trend which means that more training is probably needed to reach a perplexity (and WER) of a baseline model. As quite small amount of training data was used (because of long computational time), compared to usual amount of data used for training language models, validation set was not used to track overfitting. Thereafter, in order to check if model overfitted or not, its performance was evaluated on test set for the last three epochs. If the trend is diminishing, the model did not overfit. The results of this evaluation are presented in table 4.

Table 4: Evaluation on test set of NPLM models trained in last three epochs

Dataset	Epoch	Test set PP (pure)
train2580	13	689.764
	14	677.427
	15	667.282
train25800	13	1053.45
	14	1047.3
	15	1041.97
train258000	6	1092.34
	7	1087.32
	8	1083.59

The models did not overfit, which means that increasing the number of training epoch would be a reasonable step in order to increase the performance.

5 Discussion and Conclusions

It was shown in [12] and [8] that NPLMs trained using a neural network with one hidden layer can improve model perplexity and speech recognition performance if they trained on large enough text corpora and sufficient amount of time. The smallest corpus, used in both papers, contained 7 million words, compared to near 100000 words of the largest corpus used in this project.

For the corpora tested in this project, the speech recognizer performance became slightly worse after applying re-scoring trigram back-off model with NPLM. The main reason for this is using small

amount of data due to computational complexity of chosen NPLMs, that resulted in insufficient amount of training epochs. It turned out to be too demanding to train NPLMs using a neural network with two hidden layers (took 20 hours to train a neural network on train258000 dataset for 8 epochs only on PDC!). The performance didn't increase, but the diminishing perplexity trend is clear. It means that such models have potential to outperform n-grams given sufficient amount of training data and computational resources. However, the facts that NPLMs require significantly more time to be trained while the performance gains are not that big (if any), might prevent them for being widely adopted for speech recognition.

Nonetheless, as we live in the era of Deep learning, it might be sufficient to try a two layer fully connected network with, for instance, ReLU activation function instead of tanh. Using optimizers, different from mini-batch gradient descent, for instance Adam, might also lead to faster training of NPLMs in terms of epochs number. Another possible improvement, which was actually suggested by Bengio in [7], is using RNNs instead of feedforward networks, however training time might still be a problem.

Another disadvantage of NPLMs, in addition to training time, is the absence of isolated evaluation method. As NPLMs can't generate 1-gram models, it's hard to evaluate only NPLMs in real speech recognizer without using re-scoring procedure.

References

[1] 1 billion word language model benchmark. URL `http://www.statmt.org/lm-benchmark/`.

[2] Free speech... recognition (linux, windows and mac) - voxforge.org. `http://www.voxforge.org/`. accessed 06/25/2014.

[3] Arpa-mit lm format specification. URL `http://www1.icsi.berkeley.edu/Speech/docs/HTKBook3.2/node213_mn.html`.

[4] Neural probabilistic language model toolkit. URL `https://nlg.isi.edu/software/nplm/`.

[5] Cmusphinx open source speech recognition toolkit. URL `https://cmusphinx.github.io/`.

[6] The sri language modeling toolkit. URL `http://www.speech.sri.com/projects/srilm/`.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, Mar. 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944966`.

[8] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. *Neural Probabilistic Language Models*, pages 137–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-33486-6. doi: 10.1007/3-540-33486-6_6. URL `http://dx.doi.org/10.1007/3-540-33486-6_6`.

[9] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

[11] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[12] H. Schwenk and J.-L. Gauvain. Neural network language models for conversational speech recognition. In *INTERSPEECH*, 2004.

[13] A. Vaswani, Y. Zhao, V. Fossum, and D. Chiang. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392. Citeseer, 2013.