Report IX

Neural Probabilistic Language Modeling

Ahmad Emami and Frederick Jelinek
CLSP/Electrical and Computer Engineering

AIMS The biggest obstacle Language Modeling research has to overcome is the 'curse of dimensionality'. If we want to have a model which is capable of assigning scores (probabilities) to long sequences of words, given the vocabulary sizes in the order of tens of thousands, the model will have far too many parameters to be reliably estimated even with the largest corpora available. In this study we try to fight the 'curse of dimensionality' by its own weapon, taking the language modeling problem to a smooth continuous domain where any seen event in the training corpus will contribute to the estimation of the probabilities of its unseen neighbors.

PROGRESS The goal of statistical language modeling is to learn the joint probability function of a sequence of words in a language. This is intrinsically difficult because of the curse of dimensionality: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on N-grams obtain generalization by classifying sequences of words based on the last few words only. The work presented here fights the curse of dimensionality with its own weapons, one learns simultaneously a distributed representation for each word along with the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar to words forming an already seen sentence. The work uses neural networks for the probability function. The approach makes use of much longer contexts than the usual N-gram language models and offers improvement on them with a very different approach.

In brief, the idea of the proposed approach can be summarized as follows: Associate with each word in the vocabulary a *feature vector*. Express the joint probability *function* of word sequences in terms of the feature vectors of the words in the sequence. Learn simultaneously the word feature vectors and the parameters of the estimating function. The number of features is much smaller than the vocabulary size making it possible to reliably estimate the joint probability function. The probability function is expressed as a product of conditional probabilities of the next word given the previous ones, using a multi-layer neural network in our experiments. This function has parameters that can be iteratively tuned in order to maximize the log-likelihood or a regularized criterion of the training data. The feature vectors associated with words are also learned simultaneously, but they can be initialized with some prior knowledge in order to give prior information about the words to the system or to just speed up the convergence. The idea here is that the words which are close to each other would have similar (close) feature vectors and since the probability function is a smooth function of these feature values, so a small change in the features would only induce a small change in the probability.

More Detail The probability function $f(w_t, ..., w_{t-n}) = P(w_t|w_1^{t-1})$ is determined in two parts:

1. A mapping C of words in vocabulary V to real vectors
2. The probability function over words' feature vectors. A function g maps a sequence of feature vectors in context $(C(w_{t-n}), ..., C(w_{t-1}))$ to a probability distribution over words in V. It

is a vector function whose i-th element estimates the probability $P(w_t|w_1^{t-1})$. The *soft-max* function is used in the output layer of the neural net to make sure probabilities sum up to 1: $P(w_t|w_1^{t-1}) = \frac{e^{h_i}}{\sum_j e^{h_j}}$ where h_j is the neural network output score for word i.

Training is achieved by looking for parameters Ω of the neural network and the values of feature matrix C that maximize the penalized log-likelihood of the training corpus:

$$L = \frac{1}{T} \sum_t log P_{w_t}(C(w_{t-n}), ..., C(w_{t-1}); \Omega) + R(C, \Omega)$$

where T is a regularization term.

In the above model the number of free parameters *scales only linearly* with the vocabulary size $|V|$, and it also *scales only linearly* with the number of words in the input context as opposed to exponential scaling for state of the art *n-gram* models.

EXPERIMENTS & RESULTS We carried out perplexity and word error rate experiments on a set of different corpora. The first experiment was on Brown Corpus which is small corpus of varied English text. A vocabulary of the most frequent 16,383 was chosen. The training, validation, and test text sizes were 800000, 200000, and 181041 words respectively. A neural network with 100 hidden units was trained on the training data only. There were 4 words in the history (5gram model) and each word was distributionally represented by a 30 element vector. The network was trained for 55 iterations. The results are given in Table 1. The baseline model is a regular word based bucketed interpolated 3gram model. The reason that 5grams were not used for the baseline was that we wouldn't have gained much doing so specially on a small corpus like Brown. As can be seen in the Table the neural network improves on the regular 3gram significantly. NN2 is just like NN1 except that the initialization was done differently and that weight decay was different. Clearly some overtraining can be observed in NN1.

The second experiment was on the UPENN section of the Wall Street Journal corpus. Here the vocabulary size was 10,001 and training, validation, and test data sizes were 971631, 77130, and 86192 words respectively. Again a neural network was trained with 4 words in the history (5gram model), 100 hidden units and 30 feature words. The results are given in Table 2. Here a variety of regular word n-gram backoff models are tried as baseline. The best result is obtained by a 5gram Knesser-Ney Interpolated backoff model. The neural network by itself cannot improve on the 5gram but it's better than the 3gram model. However when we simply interpolated the neural network with the 5gram model we get the best perplexity of 121 and currently that's one of the best perplexities reported on the UPENN corpus regardless of the complexity of the model used.

The last experiment was to run a word error rate perplexity on the Wall Street Journal Speech corpus by means of nbest rescoring. Here the vocabulary size was 19007. Because the training procedure is slow, it was decided to train only on 19 million words of the training data instead of the whole 45 million words. Lattice and nbest lists were produced separately by running a recognizer on the test set. The language model part of the recognizer was a regular 3gram backoff model trained on the whole 45 million words training data. The results are given in Table 3. The baseline word error rate is 13.7%. Again all neural network models were trained with 4 words in the history and with 100 hidden units and 30 element word features. The first neural network was trained on the full vocabulary and for 6 iterations. The second one had an output vocabulary of only the 4000 most frequent words. The input vocabulary remains unchanged, ie. all the words in the vocabulary can predict only a 4000 words subset of the vocabulary. Since almost all the computations are in the output layer, we save by a factor of 5 limiting the output vocabulary.

model	train ppl	valid ppl	test ppl
3gram intp			**336**
NN1	136	289	272
NN2	189	272	**257**

Table 1: Brown corpus perplexity experiments

model	train ppl	valid ppl	test ppl
3gram KN	26	171	164
3gram KN-intp	20	158	148
5gram KN	26	177	174
5gram KN-intp	8	148	**141**
NN	131	164	157
NN+5gm			**121**

Table 2: UPENN section perplexity experiments

This network was trained for 28 iterations. All the results given for neural networks were after interpolating with a 5gram backoff model trained on the 19 million words training data and with the lattice language model scores. As can be seen the neural network improves on the WER and also limiting the output vocabulary doesn't reduce the performance while saving on the computations significantly.

FUTURE WORK We plan to further improve this method by using more input features. The neural network is a very appropriate model for this purpose in the sense that adding more inputs only increases the model linearly and that's only for a small part of the model (input layer), so actually the model grows sublinearly. Also the inputs don't have to be words and don't have to be left to right. This leaves the door open for experimenting with different configurations. The first application we are considering is to use it as the predictor in the Structured Language Model, where the inputs can be preceding words, heardwords, and non-terminal tags.

model	test WER
baseline	**13.7%**
NN-full vocab	**13.0%**
NN-limited vocab	**13.0%**

Table 3: WSJ word error rate experiments