HW2: Language modeling

Noah Golowich and Jesse Zhang

February 14, 2018

1 Introduction

In this assignment we explore techniques to perform word-level statistical language modeling. Our aim is to predict the next word in a sentence, given all previous words. We report results on a baseline trigram model, the neural network language model of Bengio et al. (2003), and a long-short term memory recurrent neural network Zaremba et al. (2014).

We then propose several extensions to these models:

- We experiment with smoothing to the counts used in our trigram model.
- We experiment with using pre-trained word embeddings as inputs to the neural network language model and the LSTM above.
- We experiment with different strategies for learning rate decay, dependent on the epoch as the LSTM trains.
- We experiment with tying the weights of the linear output (decoder) layer and the embedding (encoder) layer together, as in Press and Wolf (2016).

2 Problem Description

We view language as one long string of words, $w_1, w_2, \ldots, w_T$. Our goal is to come up with a *statistical model* Bengio et al. (2003) of language, for which, by standard rules of conditional probability, it is sufficient to have a model of the form

$$P(w_t|w_1^{t-1}) = f(w_1^{t-1}),$$

where w_i^j means the tuple $(w_i, \ldots, w_j)$, for $i \leq j$. It is impractical to have our conditional estimate of w_t depend on all of the previous words so in practice our models will have the form $P(w_t|w_1^{t-1}) = f(w_{t-n+1}^{t-1})$ for some positive integer n.

We use a fixed vocabulary $\mathcal{V}$ of size 10000; the special tags $\langle eos \rangle$, denoting end-of-sentence, and $\langle unk \rangle$, denoting an uncommon word, are included in this vocabulary. We will measure the accuracy of our language model using the *perplexity*. More specifically, the empirical perplexity on a dataset $w_1, \ldots, w_T$ for the model P is defined as:

$$\text{PPL}(P) := \exp\left(\frac{1}{T}\sum_{t=1}^{T} \log P(w_t|w_1^{t-1})\right). \tag{1}$$

This is an empirical (finite) approximation to the "true" perplexity of language, which is defined as $\exp(\mathbb{E}_y[\log P(y_t|y_1^{t-1})])$. Our goal is to minimize (empirical) perplexity of our model, as evaluated on a held-out testing set.

3 Model and Algorithms

3.1 Trigram model

In this section we model

$$P(w_t|w_1^{t-1}) = \alpha_1 p(w_t|w_{t-1}, w_{t-2}) + \alpha_2 p(w_t|w_{t-1}) + \alpha_3 p(w_t),$$

where $\alpha_1, \alpha_2, \alpha_3$ are constants determined by grid search that sum to 1, and $p(w_t|w_{t-1}, w_{t-2}), p(w_t|w_{t-1}), p(w_t)$ are determined by empirical frequencies (counts) on the training set. In particular, this model is equivalent to modeling each of $p(w_t|w_{t-1}, w_{t-2}), p(w_t|w_{t-1}), p(w_t)$ as a categorical distribution (where w_{t-1}, w_{t-2} are fixed for the first, and w_{t-1} is fixed for the second), using a uniform prior, and performing a maximum likelihood estimation.

3.1.1 Hyperparameters and extensions

We chose the parameters $\alpha_1, \alpha_2, \alpha_3$ via a grid search with resolution 0.1 on the validation set. We find that choosing $\alpha_1 = 0.2, \alpha_2 = 0.5$, and $\alpha_3 = 0.3$ performs best.

We also experimented with adding smoothing, (equivalent to performing a MAP estimate with the appropriate Dirichlet prior) to the counts which determine each of $p(w_t|w_{t-1}, w_{t-2}), p(w_t|w_{t-1}), p(w_t)$. Even adding 1 to all the counts which determine $p(w_t|w_{t-1}, w_{t-2}), p(w_t|w_{t-1})$ significantly decreased the PPL, likely because for a given w_{t-1}, w_{t-2}, there are only a very small number of words which are likely to follow them, and the data is sparse enough that adding smoothing significantly decreases the estimated likelihood of seeing these words. However, if we initialize all counts that determine $p(w_t)$ to 1, we observed a slight decrease (of approximately 1) in validation PPL; this model is the one reported in Table 1.

3.2 Neural network language model

In this section we describe our implementation of the neural net language model in Bengio et al. (2003). We pick a parameter n, called the *kernel length*, and model

$$P(w_t|w_{t-n}^{t-1}) = f(w_t; w_{t-n}, \ldots, w_{t-1}),$$

for the following function f:

$$f(w_t; w_{t-n}, \ldots, w_{t-1}) = Softmax(w_t; b + Wx + U \tanh(d + Hx)),$$

where

$$x = W^{(E)}[w_{t-n}, \ldots, w_{t-1}]$$

is the concatenation of columns $w_{t-n}, \ldots, w_{t-1}$ of the *embeddings matrix* $W^{(E)}$, reshaped to a vector.

The matrix W, which represents direct connections to the outputs, is optionally set to 0 (see hyperparameters section).

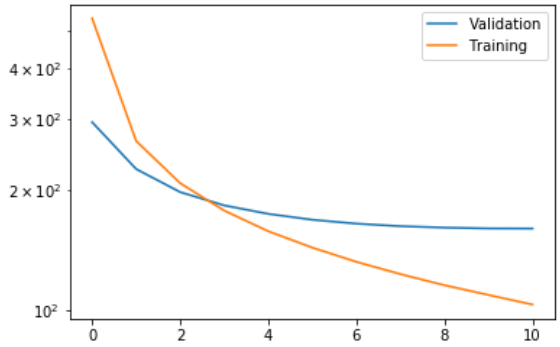

Figure 1: Plot of Validation and training PPL over the course of training. (The x-axis shows the epoch number, while the y-axis shows PPL.) We stop training when validation PPL reaches a minimum.

3.2.1 Hyperparameters

To make our batching code consistent with that of the LSTM in the following section, we train the NNLM in batches of some size B, where each batch consists of B sequences of some word length L. If we let $w_1, \ldots, w_L$ denote a single sequence (of B total) in a batch, then to evaluate $f(w_t; w_{t-n}, \ldots, w_{t-1})$ for $t < n$, we insert 0-padding for all missing words; that is, we set $w_j = 0$ for $j < 0$. It is possible to obtain performance improvements by using the last n words from the previous batch (which come directly before the words from the current batch). However, this improvement is slight for large L, and anyway the NNLM is highly outperformed by the LSTM.

After some experimentation we found that $L = 1, B = 1000$ performs best. Apart from this choice, we generally attempt to mimic the hyperparameters of Bengio et al. (2003), with variations when they lead to decreases in validation perplexity. In particular, we use the Adam optimizer with a base learning rate of $\epsilon_0 = 1.0$, which is decayed according to the formula $\epsilon_e = \epsilon_0/(1 + 0.1 \cdot \max(0, e - 6))$, where e denotes the current epoch.[1] (Bengio et al. (2003) decays parameters via a similar inverse linear formula, where the epoch number e is replaced by the number of parameter updates, and where the $e - 6$ is replaced by e. Doing so leads to a similar validation PPL but leads to a longer convergence time). We use an embedding size of 100 dimensions, a kernel size of $n = 4$, and dropout with probability of $p = 0.25$ on the outputs of the convolutional layer (that is, before the linear layer U is applied). We note that dropout was not used in Bengio et al. (2003), as the paper predates the rise in popularity of dropout. Our final model also sets the direct connections W to 0. Parameters are initialized uniformly in $[-0.1, 0.1]$. During training we ensure that the ℓ_∞ norm of the parameters in the embedding matrix never exceeds 10. We train for a total of 10 epochs, where this number was chosen via early stopping on a held-out validation set. We show in Figure 1 a plot of the validation and training PPL over the course of training.

[1] When reporting this learning rate, we average over all words in a single sequence, which is different from the following section for the LSTM, where we take a sum instead.

3.3 Recurrent neural network (LSTM)

Finally, we detail our implementation of the LSTM language model in Zaremba et al. (2014). The model is characterized by

$$P(w_t|w_{t-n}^{t-1}) = f(w_t; w_{t-n}, \ldots, w_{t-1}),$$

where the function f is descibed by

$$f(w_t; w_{t-n}, \ldots, w_{t-1}) = Softmax(w_t; b + Wh),$$

with h the final hidden state of the LSTM. The input to the LSTM at each time t is the column of an embedding matrix $W^{(E)}$ of trainable parameters corresponding to word w_t. As in Zaremba et al. (2014), we apply dropout to all "vertical connections" of the LSTM: in particular, we apply dropout to the output of the embedding matrix, between consecutive layers inside the LSTM, and to the outputs h of the LSTM (before the final linear layer W).

3.3.1 Hyperparameters

Our implementation is modeled entirely off the parameters from Zaremba et al. (2014), with regularization using a dropout of 0.5. The size of the word embeddings to be trained is not specified, but we used 100. We use 2 hidden layers, each of size 650, and a learning rate of 1, with a decay of 0.5 after the 6 epochs. We clip the ℓ_2 norm of gradients to 5 and additionally ensure that the ℓ_∞ norm of the parameters corresponding to the embedding matrix is at most 10.

We use batches of size 128, each containing sequences of size 36. (We note that Zaremba et al. (2014) uses smaller batches, but a larger batch size speeds up training due to parallelization on the GPU.) We use early stopping based on a held-out validation set; we train for a total of 50 epochs. Parameters are initialized uniformly in $[-0.05, 0.05]$.

During training, we make use of the sequential nature of batches: in particular, the final cell memory and hidden states of each previous iteration are used as the initial cell memory and hidden states of the next iteration. The initial cell memory and hidden states of the first iteration of each epoch are initialized to 0.

For the Kaggle test data, we simply applied the trained model to each sentence one at a time, effectively utilizing a batch size of 1. The sentence lengths for the test set was consistently 10, and the top 20 words (excluding $<$ `eos` $>$), were chosen to the predictions.

3.4 Extensions

Our first extension was to use pretrained word embeddings, an aspect of the model not specified by Zaremba et al. (2014). Using the pretrained word2vec embeddings of size 300, we trained another RNN model, called LSTM2, with the all other hyperparameters held constant.

Moreover, we dove deeper into the learning rate of the LSTM, which found (not surprisingly) to have a major impact on the validation PPL. To start, we held the learning rate constant at 1, throughout training, and noticed that while the PPL did descend quite quickly, it abruptly stopped going down at around 135, likely due to the gradient jumping out of a local optimum.

Next, we tried a constant decay on the learning rate, starting at 1 on epoch 0, and decreasing monotonically, according to

$$\eta = \frac{1}{1 + 0.5x}$$

for epoch x. We tried several values for the decay rate 0.5, but ultimately, this caused the PPL to flatten out too early. Our best results came from implementing a decay that caused the learning rate to decrease, starting in later epochs. That is,

$$\eta = \frac{1}{1 + 0.5(x - 6)} \; \forall \; x > 6$$
$$\eta = 1 \; \forall \; x \leq 6$$

This was the strategy we eventually used in LSTM2.

Next, we experimented with tying the weights of the word embeddings to the weights of the output linear layer (i.e. setting their parameters to be the same), as in Press and Wolf (2016); noa (2018). The intuition behind this strategy is that it allows the model to share parameters for the embeddings matrix and the linear (decoder) layer, which perform conceptually similar tasks. In particular, we can increase the size of the embeddings matrix to equal the number of parameters, but decrease the number of parameters overall. We implemented this idea in a third RNN, called LSTM3, which used an word embedding of size 650, along with a corresponding hidden layer of size 650. After some tuning, all other parameters were kept the same, except that the learning rate decay was decreased a little to 0.4. This strategy helped training a lot, significantly lowering the validation PPL.

Finally, with LSTM3, we lowered the batch size from the original 128 to 32, a value closer to that used in Zaremba et al. (2014). While a smaller batch size did indeed increase the training time for each epoch, the increase in stochasticity yielded even stronger validation results; it has previously been reported in the literature (e.g. Keskar et al. (2016)) that using smaller batches can yield to superior generalization.

4 Experiments

We present experimental results based on the testing/validation set of the Penn Treebank (Marcus et al. (1994)). Table 1 shows the PPL, as defined in (1) on this set.

Our best model (LSTM3) is able to achieve a validation PPL slightly better than the Medium Regularized LSTM in Zaremba et al. (2014).

Model	PPL
TRIGRAM	191.3
NNLM	159.8
LSTM	104.5
LSTM2	101.3
LSTM3	82.4

Table 1: Table with the main results.

5 Conclusion

We have implemented a trigram model, a single convolutional-layer neural net language model, and a long-short term memory recurrent neural network to perform language modeling. The LSTM performs best (i.e. has lowest perplexity), followed by the NNLM and then the trigram model. To further improve the LSTM to state-of-the-art (e.g. Zaremba et al. (2014) achieve of 78.4 on a large lSTM), we could try shuffling batches, varying the number of embedding features we use, and especially increasing the size of each hidden layer (the Large LSTM in Zaremba et al. (2014), which achieves 78.4, has 1500 hidden units per layer).

References

(2018). A set of examples around pytorch in Vision, Text, Reinforcement Learning, etc. original-date: 2016-08-24T03:12:48Z.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 [cs, math]*. arXiv: 1609.04836.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

Press, O. and Wolf, L. (2016). Using the Output Embedding to Improve Language Models. *arXiv:1608.05859 [cs]*. arXiv: 1608.05859.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent Neural Network Regularization. *arXiv:1409.2329 [cs]*. arXiv: 1409.2329.