NLP 201: Assignment 2

Ethan Sin

November 30, 2023

1 Introduction

For this assignment, we are given the task of language modeling with different deep learning techniques we have learned about in class. Language modeling is the concept of assigning probabilities to sequences of words (Jurafsky and Martin, 2019), and for our task we are using our models to learn a probability distribution for what next token given a sequence will be the most likely, and using that probability distribution to predict the next token. Language modeling is an unsupervised learning task as we are not working with data that is given a label, and what we are using our models to predict is the next token within the data itself. Our dataset is the Penn Treebank text-only dataset. For this dataset, most of the preprocessing is already done and split into train, validation, and test sets. The train set consists of about 42,100 sentences, the validation with about 3,370 sentences, and test set with about 3,760 sentences.

2 Models

For the models that we are using, most of the code is taken from the starter code given. The starter code is built to train a recurrent neural network (RNN) model. Adjustments were made to get the code to run properly on our local machines and Google Colab, and other additions included creating an evaluation function, and a way to track the models improvement in loss and perplexity through the epochs. Our long short-term memory (LSTM) model is also adapted from the starter code, with the addition of a cell state initialization for the model to work properly.

2.1 RNN

For our task, the RNN takes a sequence and learns by looking at each token sequentially and the next token as the desired output(Kostadinov, 2017). It will make a prediction based on the initialized weights and calculate the difference between that prediction and the actual next token by negative log likelihood loss. The loss is then passed back through the sequence of tokens and the tokens' weights are updated based on the gradient calculated with this loss. The weights that are updated during training are word embeddings associated with each individual word. Ideally, the distribution of weights in these embeddings should allow the model to make more generalizations with words that are similar and expect similar tokens after itself. If our model has pretrained embeddings active, then it will use the GloVe (Pennington et al., 2014) word embeddings. The total length of the word embedding dimension will be the size of these pretrained embeddings times two, in our case being 600 total as the embedding has a length of 300. The first 300 are frozen to maintain the value of these pretrained embeddings, the latter are initialized similarly to the frozen ones but are trainable by the model. Without pretrained embeddings active, the embeddings will simply be initialized randomly.

An issue with RNN is that the gradient calculated as it goes back farther down the network is greatly diminished. The model will then miss out on potentially very valuable information earlier on in the sequence as those weights will not be updated with as much efficacy. There are a few ways to address this issue, and the one we will be implementing is the LSTM model.

2.2 LSTM

The long short-term memory network is an RNN with gates using mathematical functions under the hood that filter the information that is passed through the network to choose what will be remembered during training (Dolphin, 2020). This should make the issue of the vanishing gradient more manageable by attempting to make the more important weights in the sequence update properly. This utilizes a cell state and hidden state that are both updated by each other in intertwining various processes.

2.3 Hyperparameters

As the LSTM is just an RNN with this gate process applied, the hyperparameters are mostly the same. These consist of our embeddings, dropout percentage, the size of our hidden layers, the size of our batches, how many layers we use, and whether or not we implement a bidirectional RNN.

3 Experiments

For our experimentation, we trained our RNN and LSTM models symmetrically to simultaneously observe the difference in performance between the models and to gather data on how the hyperparameters we train affect the data. For both RNN and LSTM, we tested 4 configurations. First is the default hyperparameters, which has pretrained embeddings inactive, dropout percentage at 0.5, and is not bidirectional. The second is with dropout percentage down to 0.1. The third is back to 0.5 for dropout but with pretrained embeddings active. The last is the same as the previous but with a bidirectional RNN. As training was more time consuming with these models, these few configurations allowed me to get some information on the effects of dropout, pretrained embeddings, a bidirectional RNN, and an LSTM in only several rounds of training and evaluation. These are all done at 30 epochs, except for a few where the performance was exceptional and it reached near 1 perplexity very early on in training. This will be discussed further.

4 Results

Configuration	RNN	LSTM
Base	37.10	31.70
Dropout 0.1	54.81	111.98
Pretrained Embeddings	35.02	43.58
Bidirectional	1.02	1.03

Table 1: Perplexities

The results ended up being much less straightforward than we expected. The base configuration is to be expected, fairly reasonable performance with the LSTM being notably better. When dropout is down to 0.1, the LSTM begins to really suffer. The model is clearly overfitting, as for example the LSTM would be reaching perplexities of 5.16 during training. Perhaps LSTMs are more prone to overfitting with low dropout as it learns from more specific information during training. A more confusing result is the performance of the LSTM becoming worse with pretrained embeddings active. The issue is also clearly overfitting, as the RNN with this configuration reaches a train perplexity of 24.76 while the LSTM reaches 11.65. A quality of the frozen pretrained embeddings seems to affect LSTMs more, and with further experimentation a higher dropout percentage may solve this and allow the LSTM to perform better.

The most unexpected result of the experiment is the activation of the bidirectional model. A bidirectional RNN traditionally will look at both sides of the output token, seeing what comes before and after. This gives a lot more information to the model, but is not our task. Our task is to predict the next token, and a normal bidirectional model would not be possible because we should not know what comes after the next token. However, we tested this configuration under the assumption that the bidirectional model was implemented to simply look at the tokens in the sequence both ways to account for the loss in information from vanishing gradients. It is still unclear to us how the model is working, but looking at the exceptional performances of the model, there is a suspicion that the model is looking ahead at tokens past the token to be predicted. It is also worth noting that during training, the bidirectional models approached 1 perplexity on the train set extremely quickly within a few epochs.

5 Conclusions

Observing the base configuration, it seems that an LSTM should perform better at this task as suspected. However, the few configurations we decided to test with seem to show that LSTMs are more prone to overfitting than a simple RNN. The amount of data we have is too small to make a proper conclusion on this, however. With more time, further experimentation on methods to reduce overfitting would be done. We hypothesize that the findings would show that LSTMs have a higher potential performance for this task with the right configurations, as even the base configuration we have is the best performance not counting the bidirectional model. We are also very hesitant to make any conclusion about our models on this task using the bidirectional model as it seems it may be incorrectly performing the task to produce extremely desirable results.

References

Dolphin, R. (2020). Lstm networks | a detailed explanation. *Towards Data Science*.

Jurafsky, D. and Martin, J. H. (2019). *Speech and Language Processing*. Stanford University Press.

Kostadinov, S. (2017). How recurrent neural networks work. *Towards Data Science*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.