Final
CMPT469 Deep Learning w/ TensorFlow

Kai Wong

12/11/2017

1 Experiments and Analysis

1.1 Dataset Chosen

For this assignment, I chose the PTB Dataset, which consists of 10k rows of general-purpose sentences. I wanted to work with general purpose sentences so that I would have more words to work with in the final experiments. However, I soon regretted working with those 10k entries and wished I had worked with the Mark Dataset instead, which I learned later consists of merely 2k entries. This would have made getting results feasible given the hardware I had access to.

1.2 Adding Forget Bias

Adding Forget Bias

	0 Forget Bias	1 Forget Bias	2.5 Forget Bias	5 Forget Bias
Test Perplexity	112.154	105.854	117.981	131.577

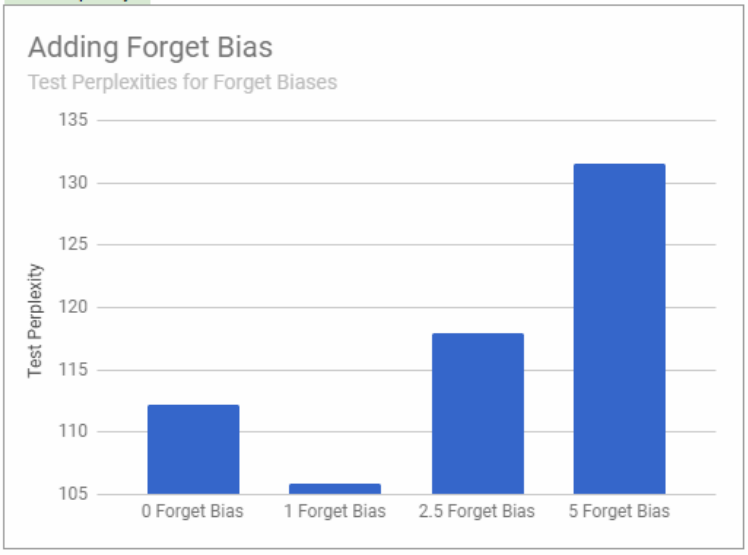

Figure 1: Adding a Forget Bias to the RNN-LSTM Network
Performed with a Learning Rate of 1.0, Decay of 0.25, 200 Neurons

The original RNN-LSTM network did not have a forget bias in its LSTM cells. Past works have strongly indicated that adding a forget bias of 1 to the LSTM's forget gate greatly improves the performance of the RNN-LSTM network as a whole. Putting a forget bias of 1 as shown in Figure 1 for this RNN-LSTM network greatly improved its performance over a forget bias of 0. LSTM cells have forget gates that allow them to forget information that may not be relevant for them to know anymore for future predictions. For

example, once we reach the end of a sentence, we do not have much reason to believe the next sentence will have anything to do with that previous sentence. Increasing the forget bias too much, however, increased perplexity, and this shows that the cells were probably forgetting too much.

From this, we can conclude that a value of 1.0 is the best value to set for the forget bias.

1.3 Adding Dropout

Adding Dropout For Different Neuron Sizes

	1.0 Dropout	0.9 Dropout	0.8 Dropout
200 neurons (Train)	54.349	77.105	89.892
200 neurons (Validation)	116.521	112.251	112.263
200 neurons (Test)	112.154	107.878	107.509
400 neurons (Train)	35.67	54.672	68.891
400 neurons (Validation)	118.921	103.99	102.963
400 neurons (Test)	114.608	100.227	98.317

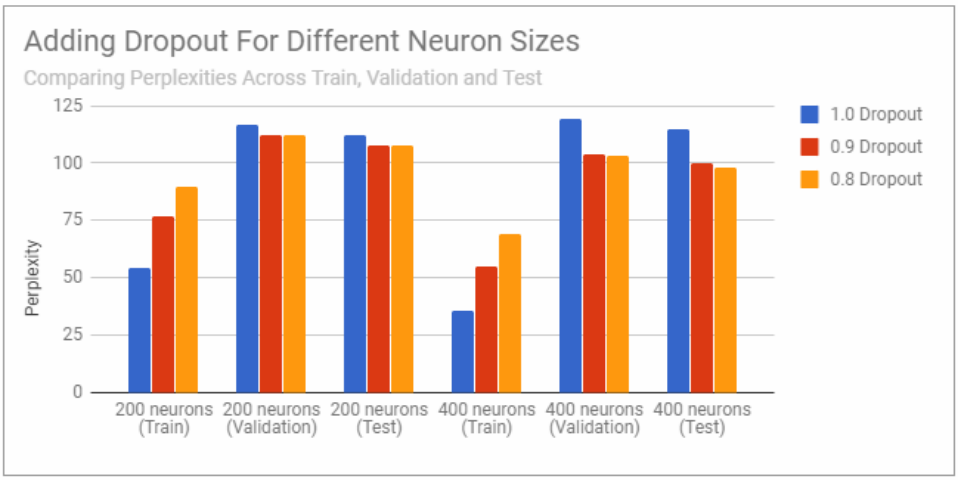

Figure 2: Adding a Dropout For Different Neuron Sizes to the RNN-LSTM Network Performed with a Learning Rate of 1.0, Decay of 0.25, Forget Bias of 1.0

Having a dropout layer in a network introduces a form of regularization. This is important because this reduces overfitting and improves the model's performance in generalization. I noticed that this network was prone to overfitting, as LSTMs in general easily overfit training data. Later in Figure 5, we can see the model overfitting, for when we increase the number of hidden units, the validation and testing perplexity increases, despite a steep decrease in the training perplexity. This is a sign of overfitting, where the model learns the data too well in training, and as a result, fails to generalize well.

By adding dropout, we can see in Figure 2 that final testing perplexity decreases across both 200 and 400 neurons in validation and testing as we increase the amount of dropout (from 1.0 (none) to 0.8). We can also notice that the final training perplexity increases as we increase the amount of dropout. This shows the effects of dropout; it is constraining network adaptation to the data at training time, preventing it from learning the input data too well, thus helping to avoid overfitting. Thus, the training perplexity increases as we add more constraint.

We can also see the effect that dropout has against overfitting. As stated earlier, the RNN-LSTM runs into a huge problem of overfitting on increased number of hidden units. However, as we can see in Figure 2 with an increased hidden unit size of 400, increasing the dropout causes the validation and testing perplexity to come far down, and increases the training perplexity significantly. This shows the impact that overfitting

has on the network, and how dropout successfully reduces the effect that it has on model performance.

From this, we can conclude that a value of 0.8 is the best value to set for the dropout.

1.4 Varying Sequence Length

Varying Sequence Length

Number of Neurons:	20 seq test perplexity	25 seq test perplexity	30 seq test perplexity	40 seq test perplexity
200 neuron	117.285	117.636	116.741	117.095
400 neuron	132.833	127.357	130.956	125.976
200 neuron	114.833		117.695	

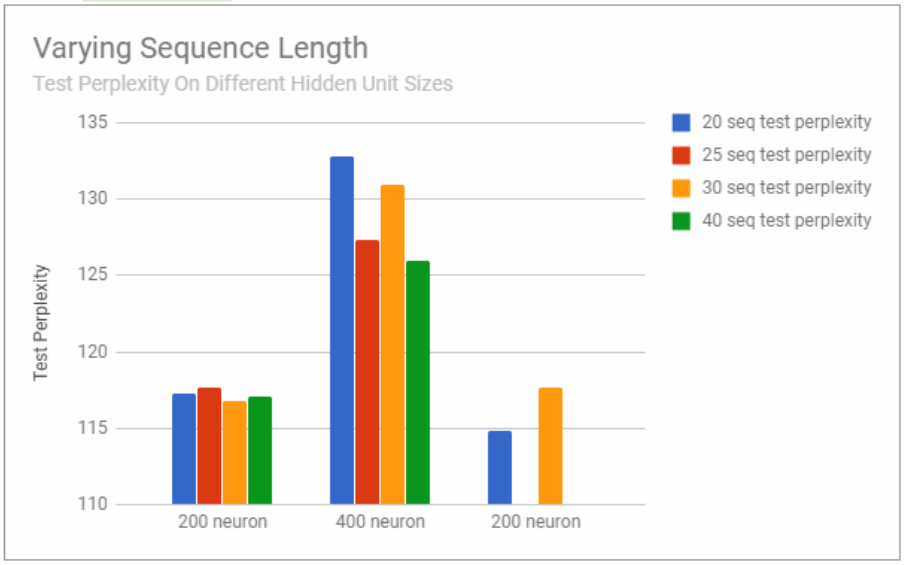

Figure 3: Varying Sequence Length on Different Neuron Sizes
Performed with a Learning Rate of 1.0 and Decay of 0.5

Having a good sequence is important in an RNN-LSTM network that is built to perform language modeling. We need to give enough context to the model for it to train on, and we do this by varying the sequence length, or how many words it sees at a time during training. According to Google, the average sentence length is about 20 words. Therefore, a good sequence length should be about 20. Tests were performed on increased sequence lengths, but this did not seem to have much or any correlation to the final testing perplexity, as can be seen in Figure 3. In some cases, it increased testing perplexity. This probably meant that the network was being given too much context to learn from; too much context is not a good thing, for a word from the beginning of a sentence or an earlier sentence won't necessarily help us to predict the next word later in the sentence or in the next sentence.

From this, we can conclude that a value of 20 is the best value to set for the sequence length.

1.5 How many epochs?

Epochs Until Reach Limit of Training

Epoch	Train Perplexity	Valid Perplexity
0	373.21	215.371
3	137.477	144.599
6	85.442	114.23
9	74.832	110.09
12	74.503	109.961
15	74.538	109.96
18	74.556	109.96
21	74.599	109.96
24	74.542	109.96
27	74.556	109.96
30	74.561	109.96
33	74.565	109.96

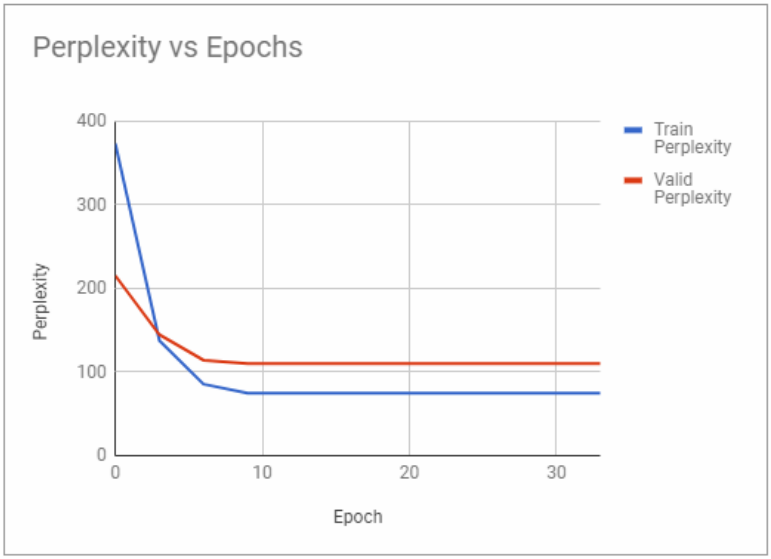

Figure 4: Perplexity vs Epochs
Performed with a Learning Rate of 1.0, Decay of 0.25, Forget Bias of 1.0, .9 Dropout

Determining the right number of epochs for the network to train on is important in figuring out how much training is needed for a network to converge. Too much training of a network will cause it to overfit, where it starts to learn the data of the input too well and validation perplexity starts to go up. In this case in Figure 4, the network's validation and testing perplexity seems to stop decreasing after about 13 epochs. For the many epochs afterwards, these perplexities do not improve; we can infer that the network is most likely overfitting here, where we are performing training that does nothing to improve the state of the network. Therefore, it would be wise of us to stop training after this many epochs.

From this, we can conclude that a value between 13 and 18 is the best value to set for the number of epochs.

1.6 Varying Hidden Units

Varying Hidden Units with 1.0 Learning Rate and 0.5 Decay Over 13 Epochs

	200 neurons	400 neurons	600 neurons
Train	47.055	25.662	18.049
Validation	122.345	137.224	160.12
Test	117.285	132.822	154.328

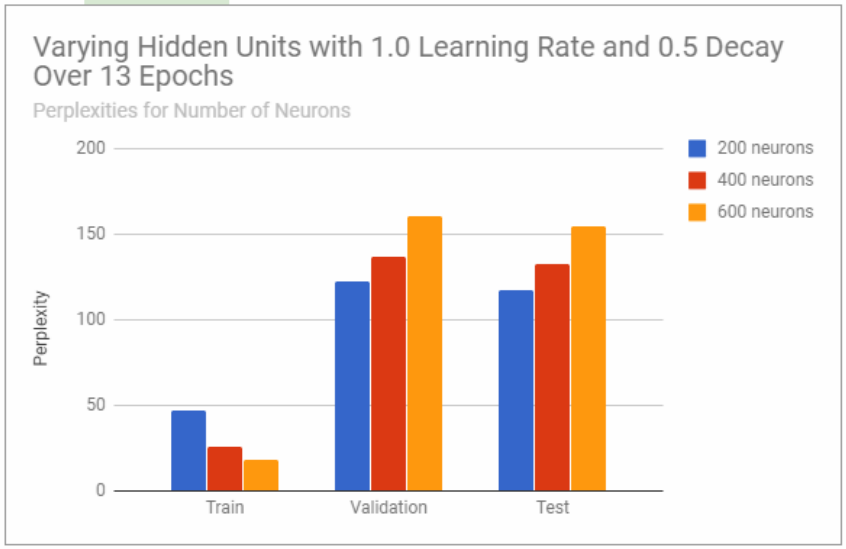

Figure 5: Varying Neuron Sizes
Performed with a Learning Rate of 1.0, Decay of 0.5

In an RNN-LSTM network, it is important to have a good number of hidden units. This is due to the hidden units directly affecting the embedding that occurs in the network, which is the most important feature in an RNN-LSTM network. This is where a word is transformed into some vectorial representation of information that the model will use in determining what should come after that word. By increasing the number of hidden units, we allow more embedding to occur (as a word is being transformed into a larger vector), causing the model to store more information about the data coming in. Of course, one would think that as a result of this, the training of the model should be more effective, as it has more detailed data to work with. However, this proves to not be the case. The opposite occurs, where the model now stores too much information about the input, and it actually ends up learning the input far too well, which causes it to overfit and fail in generalizing well to test cases. We can see this occurring in Figure 5, where when we increase neuron size, training perplexity decreases, but validation and test perplexity skyrocket. This shows how in training, the model learns the input extremely well, but in validation and testing, the model fails to generalize well. Of course, this is shown earlier in Figure 2 to be preventable by introducing a dropout layer, and by doing so, an increase in the number of neurons is ideal and brings the perplexity of the network much lower. Figure 2 also shows how effectiveness of increasing dropout when there are more hidden units; due to more hidden units and thus more overfitting, it seems necessary to have more dropout to compensate for that.

From this, we can conclude that, with a dropout layer introduced into the network, that a large hidden unit size is ideal, preferably between 400 and 600.

1.7 Varying Learning Rate and Decay for Different Neuron Sizes

Varying Learning Rate and Decay for Different Neuron Sizes

	200 neurons	400 neurons	800 neurons
Learning Rate of 1.0 and Decay of .5	117.285	132.822	154.328
Learning Rate of 1.0 and Decay of .25	112.154		
Learning Rate of 2.0 and Decay of .25	114.833	100.227	
Learning Rate of 2.5 and Decay of .25			119.91
Learning Rate of 3.0 and Decay of .25	121.617		115.259

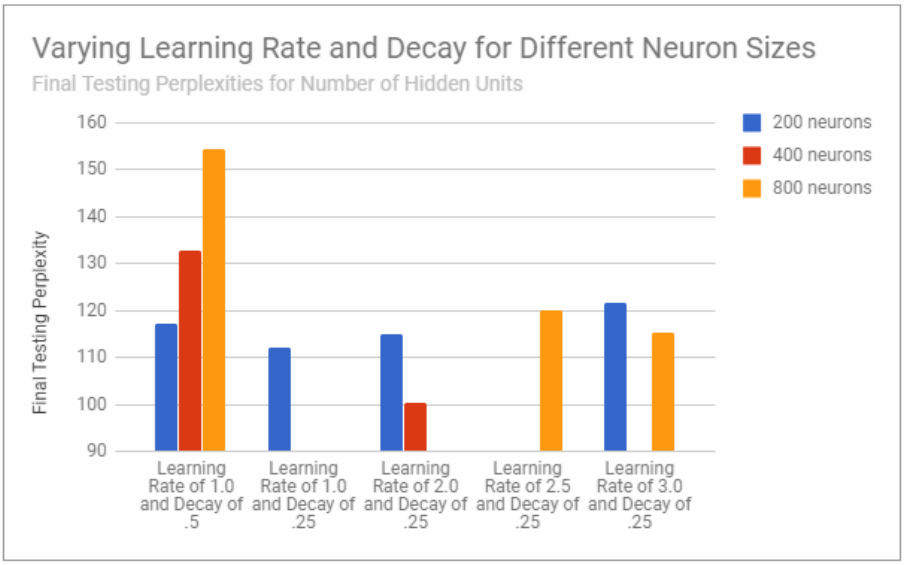

Figure 6: Varying Learning Rate and Decay for Different Neuron Sizes
Performed with a Forget Bias of 1.0

As can be inferred from the data in Figure 6, we can see that the ideal learning rate for each number of neurons in the hidden layer is directly proportional. For 200 neurons, an initial learning rate of 1.0 gives better results than higher learning rates of 2.0 and higher. For 400 neurons, an initial learning rate of 2.0 gives better results than a lower learning rate of 1.0. For 800 neurons, an initial learning rate of 3.0 gives better results than lower learning rates of 2.5 and 1.0. In general, a higher initial learning rate for more neurons will allow a model to better learn the distribution of data. Models with a larger hidden unit size will need a higher initial learning rate in over to explore more of the error function with that bigger step size. The same goes for models with a smaller hidden unit size; with fewer neurons, there is less error function to explore. Thus, the initial step size must be small enough to not continually overshoot the global minimum.

The learning rate will have to decay later during the training so that the model can have a smaller learning rate such that it can continually take smaller and smaller steps and perform a more detailed exploration of the error surface. In general, it seems that a larger decay in learning rate (0.25 vs 0.5) improves testing perplexity; this is because the model, after taking big initial steps in exploring the error function to find a region of general, large-scale optima, can start quickly decreasing its step size so that it can perform a more focused exploration of a particular optimum. The more quickly it can do this, the more accurate it will be in finding the global minimum, for it will have more smaller steps to do so.

From this, we can conclude that a learning rate of 2.0 with a decay of .25 is ideal for a hidden unit size of 400 neurons, while a learning rate of 1.0 with a decay of .25 is deal for a hidden unit size of 200 neurons.

2 Best Results

2.1 Parameters

By running the network with the following: an initial learning rate of 2.0, a decay of 0.25, a sequence length of 20, a hidden size of 400 neurons, a dropout of 0.8, a forget bias of 1.0, and a total epoch number of 13, we receive a final test perplexity of 98.317.

3 Questions

3.1 What was your strategy in the search of the best set parameters?

Having worked with another RNN-LSTM model for the final project, I noticed that the LSTM-cell had a forget-bias of 1.0, while the RNN-LSTM model given here did not have a forget-bias. Therefore, I changed the model to include a forget-bias, because I knew that the forget gate plays an important role in LSTM functionality. I also applied knowledge from previous work with RNN-LSTMs, such as in Homework 2 and the Midterm, where we varied the number of hidden units, the learning rate, and the sequence length. From these past experiments, I found that increasing the number of hidden units improved performance of the RNN-LSTM. I had also found that the initial learning rate should not be too small, so I applied the same past knowledge to these current experiments. However, I found that increasing the number of hidden units made the network overfit, which I thought was strange; after some online research, I realized that introducing a dropout layer would prove to be very effective for combating overfitting as I increased the number of neurons; as a result, I got very accurate test results with a large hidden size and a decent dropout layer. From there, I continued to vary hidden size, along with parameters such as learning rate and dropout to match the hidden size.

3.2 Based on this experience, which parameters you think are the most influential, crucial, or important to yield a good result?

The learning rate, decay, dropout, forget bias, and the hidden size proved to be the most influential parameters for yielding a low testing perplexity. To reiterate from previously in Figure 5, we know that increasing hidden size allows the embedding of the network to be more effective; this allows the model to retain more and more detailed knowledge about the input. However, this leads to overfitting, which is why dropout (Figure 2) is crucial to have in order to prevent such an occurrence. A good learning rate and decay (Figure 6) is important as well, as for any network, as this is the step size at which the optimizer traverses the error function to find the global minimum. Due to an increased neuron size, we must have a larger learning rate and decay to compensate, such that we can explore the error function effectively with large steps in the beginning epochs, and then quickly decrease those steps such that we can start exploring the error function in more detail. If a small initial learning rate is chosen with a small decay, the optimizer will fail to find the global minimum effectively in such a large error function, and moreover, if it does, a smaller decay will prevent it from shrinking down quick enough to perform a good number of detailed traversals to find the minimum better. Finally, the forget bias (Figure 1) is important to have, as forget gates in LSTMs are important to have functioning properly; otherwise, the network retains past knowledge that is no longer necessary to have.

3.3 Based on this experience, which parameters you think are the least influential, crucial, or important to yield a good result?

The number of epochs and the sequence size proved to be the least influential parameters for yielding a low testing perplexity. Varying them in various increments, other than in extremes, hardly did anything to change the perplexity. To reiterate from previously (Figure 3), sequence size does not matter too much, as long as it is large enough; enough context must be given to the model to learn from, but not too much or not too little; giving it too much context does not make a marked difference on its performance, for words very far back in sentences do not do much to help predict what the next word in a sentence will be. Of course,

if too little context is given, then the model will not learn to effectively predict the sequences of words, but this would occur with very small sequences, showing how small variations in sequence size from 20 hardly has an influential impact. The number of epochs also does not have much influence; as shown in Figure 4, after a certain number of epochs, the network's train and test perplexity do not change much. Therefore, increasing or decreasing the number of epochs by a certain amount will not do much to change or improve the test perplexity.

3.4 Observing the experiment that gave you the best results, discuss:

3.4.1 What happens to the perplexity in training, validation, and testing sets? Why do you think that is?

Epoch	Train Perplexity	Valid Perplexity
1	372.83	211.532
2	174.438	157.173
3	136.748	138.655
4	119.336	131.485
5	109.663	127.771
6	83.006	108.03
7	72.742	104.514
8	69.979	103.576
9	69.155	103.123
10	69.005	103.004
11	69.014	102.973
12	68.949	102.965
13	68.891	102.963

Final Testing Perplexity:
98.317

Parameters:
2.0 Initial Learning Rate
0.25 Learning Rate Decay
20 Sequence Length
400 Hidden Neuron Size
0.8 Dropout
1.0 Forget Bias
13 Epochs

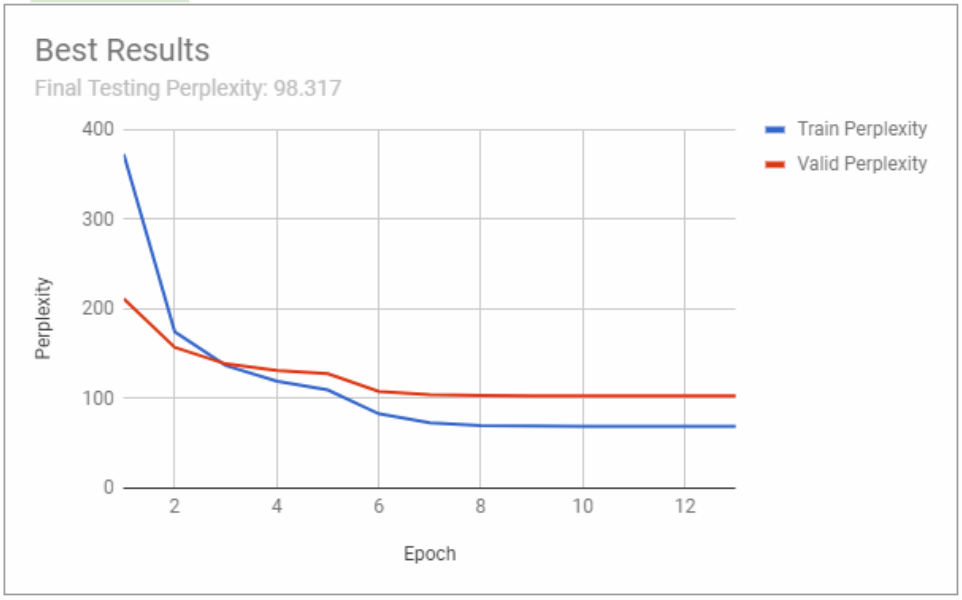

Figure 7: Best Results

The training perplexity ends at a reasonable number of 68.891, and valid perplexity at 102.963. On the graph, they both appear to flatten out after 10 or so epochs, perhaps showing that 10 epochs would have been enough testing for this version of the model. However, the data shows the train and valid perplexity going down in tiny amounts after 10 epochs; the model is still improving, even if slightly. The model's test and valid perplexity improve drastically in the first few epochs, showing the effects of the rapid initial learning rate. The final test perplexity is 98.317. Due to 400 hidden neurons, the network can store a decent amount of information about the input. Having a 0.8 dropout allows the network to not overfit due to

having more information on the input, and a forget bias helps it to forget information as the model trains. A 2.0 learning rate allows it to quickly traverse the large error function due to the 400 hidden neurons, and a 0.25 decay allows it to quickly grow smaller after a while in order to converge effectively. Finally, since the average sentence length is 20, a sequence length of 20 can be considered as a helpful factor for the final test perplexity.

3.4.2 What are your thoughts on the quality of the sentences that it produces?

After running my model with its parameters for 12+ hours, I was forced to stop its running, due to there being a class held in the lab. I had my tests running on 4 separate lab machines (see screenshot in GitHub), but they still had not finished; I was also unable to get any results from these tests, as I was redirecting Windows command line output to a text file (if the process is terminated, the file does not get written to). Therefore, I have no results with a full run of my best model.

I had a feeling that my model parameters would take an extremely long time to run. Therefore, as a precaution, I ran a version of the model that in my experiments yieled a testing perplexity of 105.854, with a smaller neuron size of 200, a learning rate of 1.0 and decay of 0.25, 0.9 dropout, and 1.0 forget bias for 13 epochs. Here is a final sentence with a primer word of "piano," which has over 8000 occurrences in the PTB dataset:

```
piano <unk> of the <unk> of the <unk> of the <unk> of the <unk>
 of the <unk> of the <unk> of
```

The quality of the sentence produced as a whole was poor, but the words somewhat made sense one after the other (with "the unk" making sense to come after "of"). As can be seen, the model repeatedly falls into the pattern of "of the <unk >", where there is not much variation in its prediction. It keeps on predicting the same pattern of words, but this is better than the model constantly predicting the same word one after another (i.e. the the the). For some reason, when producing sentences as opposed to just testing, the train and validation perplexity skyrocketed to 197.514 train perplexity and 214.952 valid perplexity (original experiment had 74.703 train perplexity, 110.965 valid perplexity). Perhaps this is why it continuously predicted the same pattern of words. I think that the skyrocketing perplexity may be related to the batch size being 1, as opposed to being 30 when the experiments on the model were being run. As we decrease the batch size, fewer samples are propagated throughout the network, which then causes the accuracy of the estimate of the gradient to become worse.

I tried another primer of "the":

```
the <unk> of the <unk> <eos> the <unk> <unk> <unk> <unk>
<unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk>
```

Once again, the train and validation perplexity skyrocketed to 194.585 and 212.870, showing how the model became a lot worse when batch size was brought down to 1. Here, the model continuously predicted the same word "unk", which is not coherent.

I had obtained a partial run of my best model on the PTB dataset, using a primer word of "piano":

```
piano <eos> the <unk> <eos> the <unk> <eos> the <unk> <eos> the <unk> <eos>
 the <unk> <eos> the <unk> <eos> the
```

As follows from above, the model is able to predict word sequences, but without much variation at all, only going between three different words.

I tried running tests on the Mark dataset in an attempt to get feasible results that I could analyze. Following the same parameters I had used for the PTB dataset model with 200 neurons, I tried a primer of "they":

```
they asked him, Are you not know that you are the Son of Man is not know that you are the
```

Here, the quality of the sentences produced were a lot better because the predictions varied between more words, and I was getting a lower train and valid perplexity of 57.381 and 200.662. This is most likely due to the parameters I ran for this model being a better fit for the Mark dataset. There were more variations in the sequences of words, such as: "they asked him, are you not" and "know that you are the Son of Man." Grammar could be better, but the job of the language model is to not have sentences as a whole make sense, but just to be able to predict the probabilities of the next word after a certain word. Therefore, we can see that the model is doing a pretty good job of it. The words make sense one after another.

I tried the same experiment on the Mark dataset, but with the parameters I found to be best on the

PTB dataset:
`they to to to to to to to to to to to to to to to to to to to`

Here, the quality of the sentence was horrible, as the model was predicting the exact same word one after another. The same word one after another hardly makes sense. I was also getting a much higher train and valid perplexity of 423.957 and 290.253.

3.4.3 Is the quality of the sentences congruent with the perplexity on the validation set?

For the two primers I tried on the PTB dataset, the model produces words that generally make sense one after another, but there was only variation between 3 words, and the perplexities were quite high. For the primer I tried on the Mark dataset, we can see that the sentences produced are of higher quality (more diverse word sequence predictions, rather than just variations between only 3 words), and we know that perplexity on the validation set was lower in the Mark dataset than in the PTB dataset. Therefore, we can speculatively conclude that sentence quality is congruent with the validation set. Word sequence prediction is more varied when validation perplexity is lower, and we know that this is because perplexity on the validation set was lower in the Mark dataset than in the PTB dataset. Therefore, the model was better at producing sentences with varied words. For tests in the Mark dataset where I got a higher validation perplexity, the model began to continuously predict the same word one after another, showing how it was failing in its prediction; but when perplexity was lower, it was able to predict next words with more variation, thus being better.

3.5 Final Notes

Please see my GitHub for all of my results.
`https://github.com/Holayn/CMPT496-Deep-Learning/tree/master/wong-final`

4 References

https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/
https://www.quora.com/How-does-the-dropout-method-work-in-deep-learning-And-why-is-it-claimed-to-be-an-effective-trick-to-improve-your-network
https://datascience.stackexchange.com/questions/410/choosing-a-learning-rate
https://stackoverflow.com/questions/34476447/should-i-increase-or-decrease-learning-rate-if-i-add-more-neurons-or-weights-to/34476671
https://deeplearning4j.org/lstm.html
http://proceedings.mlr.press/v37/jozefowicz15.pdf
https://machinelearningmastery.com/improve-deep-learning-performance/
https://www.quora.com/Intuitively-how-does-mini-batch-size-affect-the-performance-of-stochastic-gradient-descent
https://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network
https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network
http://www.marekrei.com/blog/26-things-i-learned-in-the-deep-learning-summer-school/
http://corochann.com/penn-tree-bank-ptb-dataset-introduction-1456.html
https://www.marist.edu/webapps/facultybio/view.html?uid=503