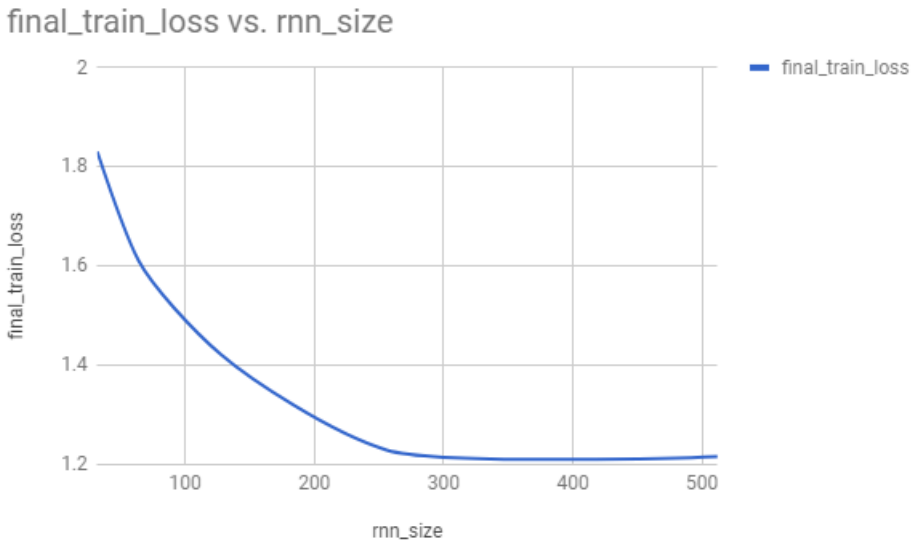
Midterm
CMPT469 Deep Learning w/ TensorFlow

Kai Wong

10/23/17

1 Part 1: Vary the number of hidden units exponentially as follows, 32, 64, 128, 256, and 512; compare results of each and answer the following questions:

rnn_size	final_train_loss
32	1.831
64	1.611
128	1.421
256	1.229
512	1.216



1.1 What happens to the perplexity? Why do you think that is?

(Note that `rnn_size` is the number of hidden units)

As we increase the number of hidden units, the perplexity of the model goes down.

```
cell = tf.contrib.rnn.BasicRNNCell(rnn_size)
```

We can see that each LSTM cell is being created with the number of hidden units (neurons). This means that in the model, when the cell is unfolded, for each of its hidden layers, the hidden state vector for that layer is of the size of the number of hidden units that we are varying.

```
loss = tf.contrib.legacy_seq2seq.sequence_loss_by_example([self.logits],
           [tf.reshape(self.targets, [-1])],
           [tf.ones([batch_size * seq_length])],
           vocab_size)
self.cost = tf.reduce_sum(loss) / batch_size / seq_length
```

As we can see in the above lines of code, this is our loss function, and our objective is to minimize the value it produces, which in turn minimizes the cost and perplexity (since less cost means a lower measure of prediction error, and more cost makes predictions harder. Cost is the error between estimated and actual output). However, how does the number of hidden units impact how well we minimize the loss/cost function?

```
softmax_w = tf.get_variable("softmax_w", [rnn_size, vocab_size]) #128x65
logits = tf.matmul(output, softmax_w) + softmax_b
```

We know that the softmax is used to predict the next character. We can see here that the softmax regression function's weight (`softmax_w`) shape is directly impacted by the number of hidden units in the cell. This, in turn, impacts the logistic unit that is used to predict the next character, and the better its prediction means we can better minimize the loss function and the perplexity, and less error is propagated within the RNN. By increasing the number of hidden units in the cell, we increase the shape size and effectiveness of the softmax's weight, thus improving the logistic unit and predictions made. The softmax's weight is important is training, because the RNN is fed back its weight (the previous state of the world) during the training process. The less hidden units, the less informative data that is being passed between each hidden layer, thus there being a less effective training. With more hidden units, we have less loss/error, thus a better training.

The number of hidden units also affects the embedding, which is the most important feature in an RNN/LSTM network; in our case, it's where a character is transformed into some vectorial representation of information that the model/network will use in determining what should come after that character.

```
embedding = tf.get_variable("embedding", [vocab_size, rnn_size])
```

Here, we can see that the embedding is directly impacted by the number of hidden units in the cell. Fewer hidden units means less embedding can occur (because a character is being transformed into a smaller vector), causing the model to store less information about the data coming in, thus the training in the model is less effective as it has less detailed data to work with, yielding a higher perplexity overall as there is bound to be more error. This phenomena of there not being enough neurons in the hidden layers for a model to learn from is known as underfitting, and it's what we see here with too few hidden units. The network does not have enough neurons to detect signals from the data coming in.

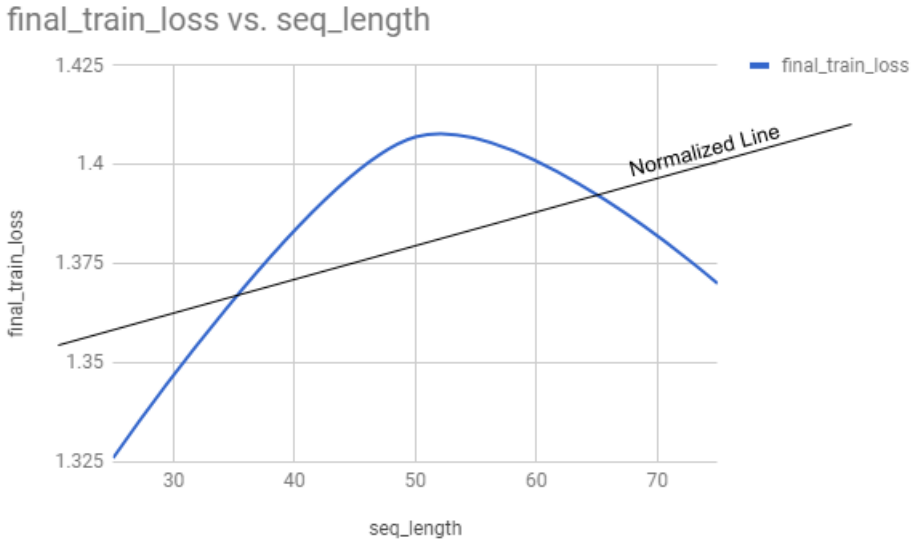
1.2 What happens to the sentences that it produces? Why do you think that is? How does this relate to the previous question about perplexity?

rnn_size	final_sentence
32	The rrow have anothoughasiy an, I conger werar'ss. fo
64	The didastany is not gentle Than being have side of Et
128	The writtle grief and out of so, To but him out And a
256	The did you mighty and all Anfears; my body, I'll plea
512	The come, Looking, what more: I am I time by they art

In terms of the sentences produced, they become more accurate as we increase the number of hidden units. This relates directly back to the model's perplexity. By increasing the effectiveness of the softmax regression function, it in turn positively impacts the logistic unit that returns the probability of the next character, which in turn minimizes the loss function and the perplexity of the model. We also know that increasing the number of hidden units allows for more embedding to occur, which allows the model to learn better and decreases its perplexity. If the model has a lower degree of perplexity, it will have a higher probability/certainty of what the next character should be, so in turn, the sentences produced should be of higher accuracy. With a better softmax function, we have the ability to better predict the next character.

2 Part 2: Vary the length of the sequences as follows, 25, 50, and 75; compare results of each and answer the following questions:

seq_length	final_train_loss
25	1.326
50	1.407
75	1.37



2.1 What happens to the perplexity? Why do you think that is?

As we increase the length of the sequences, the perplexity worsens.

```
def load_preprocessed(self, vocab_file, tensor_file):
       with open(vocab_file, 'rb') as f:
           self.chars = cPickle.load(f)
       self.vocab_size = len(self.chars)
       self.vocab = dict(zip(self.chars, range(len(self.chars))))
       self.tensor = np.load(tensor_file)
       self.num_batches = int(self.tensor.size / (self.batch_size * self.seq_length))

def create_batches(self):
    self.num_batches = int(self.tensor.size / (self.batch_size * self.seq_length))

    # When the data (tensor) is too small, let's give them a better error message
    if self.num_batches==0:
       assert False, "Not enough data. Make seq_length and batch_size small."

    self.tensor = self.tensor[:self.num_batches * self.batch_size * self.seq_length]
    xdata = self.tensor
    ydata = np.copy(self.tensor)
    ydata[:-1] = xdata[1:]
    ydata[-1] = xdata[0]
    self.x_batches = np.split(xdata.reshape(self.batch_size, -1), self.num_batches, 1)
    self.y_batches = np.split(ydata.reshape(self.batch_size, -1), self.num_batches, 1)
```

We know that sequence length directly correlates with how many hidden layers there are in the unfolded RNN. In RNNs especially, errors made early on in the training process propagate through the layers and continuously worsen over time, and the more layers there are in the RNN, the more the error propagates. We can see this here, that as we increase the sequence length, we increase the number of hidden layers in the RNN, thus causing the error/perplexity to increase because errors are propagated through the layers. However, what could be the cause of error in the first place due to varying sequence length?

As can be seen with `self.num_batches = int(self.tensor.size / (self.batch_size * self.seq_length))`, a higher sequence length causes the number of batches to be fed in each epoch to be less, meaning data is being fed fewer times into the network during training; even though with each batch, more data in the form of a longer sequence length is being fed in, neural networks in their own nature benefit from more iterations of data, whereas with more data in a batch, a significant degradation occurs (see https://arxiv.org/abs/1609.04836). Giving it too much data in one batch could cause the network's performance to suffer, as the amount of data could exceed the network's capacity to handle it effectively, and also lead to there being too much error propagation due to the increased sequence length (number of hidden layers). These factors of poor batch number and quality could cause the model to have poor performance and make more errors in training, and this is a problem especially in recurrent neural networks, as the error will propagate through layers and cause training overall to worsen, as discussed earlier. Thus, increasing sequence length gives us a higher perplexity, since the model has more error resulting it in being less certain about what the next character will be.

As a side note (not related to perplexity), when we load the data into the model with `self.tensor = np.array(list(map(self.vocab.get, data)))`, this tensor has a certain size. We determine the number of batches to run in an epoch by this size divided by the batch size and sequence length. If the sequence length gets too big, we run into the problem of there not being enough data in the data tensor to sufficiently support a longer sequence length.

2.2 What happens to the quality of the sentences that it produces? Why do you think that is? How does this relate to the previous question about perplexity?

seq_length	final_sentence
25	The cargarinan carroodless be a mending incius! lovour
50	The queen's sinesty soul as p
75	The it as my purmockises of stoars so: Then justrous Walt thy nake, that's mowk

As for the quality of the sentences, they seem to worsen as we increase the length of the sequences, and of course this happens because of the increased perplexity in the model. With an increased perplexity comes an increased margin of error in the predictions that the model makes. We know from before that by increasing the length of the sequence, we actually decrease the number and quality of the batches the model trains with, thus causing the model to become worse at predicting characters (more data in a batch = significant degradation). Furthermore, the error that is made within the model is propagated throughout the hidden layers, causing the error/perplexity to become even worse. Thus, we can see the effects of this, as the number of words that makes sense in a sentence produced by the model goes down as sequence length increases.