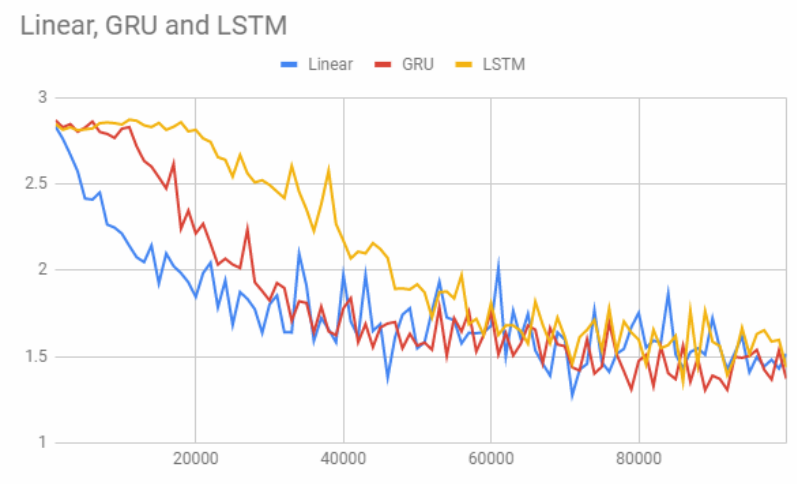
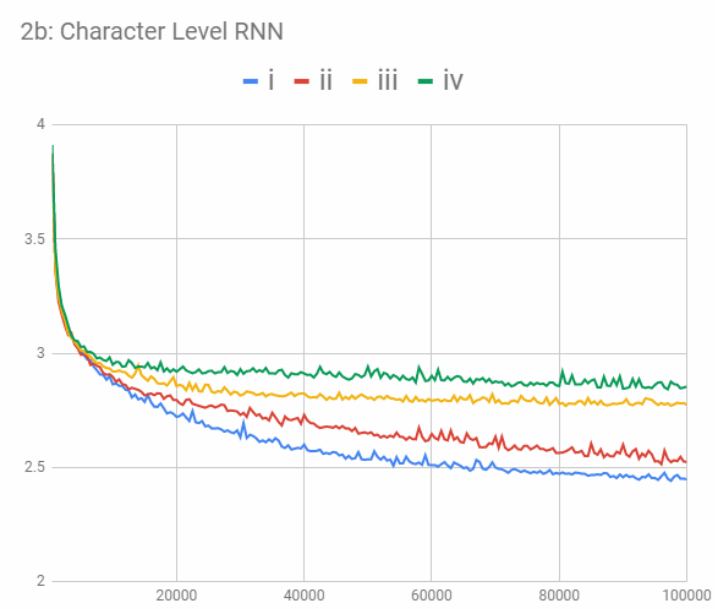
CS680 A5 Kilby Baron 20773955

1a)



GRUs are faster to train than LSTMs because there are fewer tensor operations; unlike the LSTM, the GRU has no cell state, 2 gates instead of 3, and fewer weights. Linear units are faster to train than GRUs for the same reason, there are fewer tensor operations because there are no gates. After 100,000 time steps, all 3 units are quite similar; this indicates that gradient explosion and vanishing is not as significant at the scope of individual words.

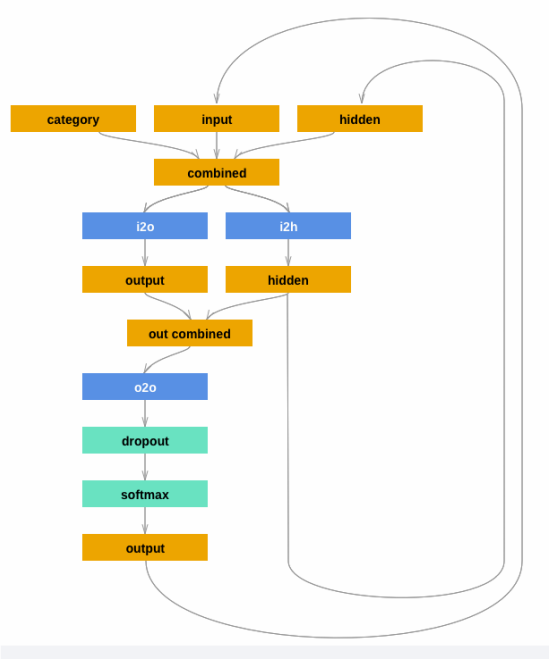
1b)



<u>Model variations:</u>

i) Previous hidden unit, previous character, and category
ii) Previous hidden unit, and previous character
iii) Previous hidden unit and category
iv) Previous hidden unit only

Question 1b uses a small RNN with a few linear layers to generate names from different languages. The RNN predicts names by outputting one letter at a time.

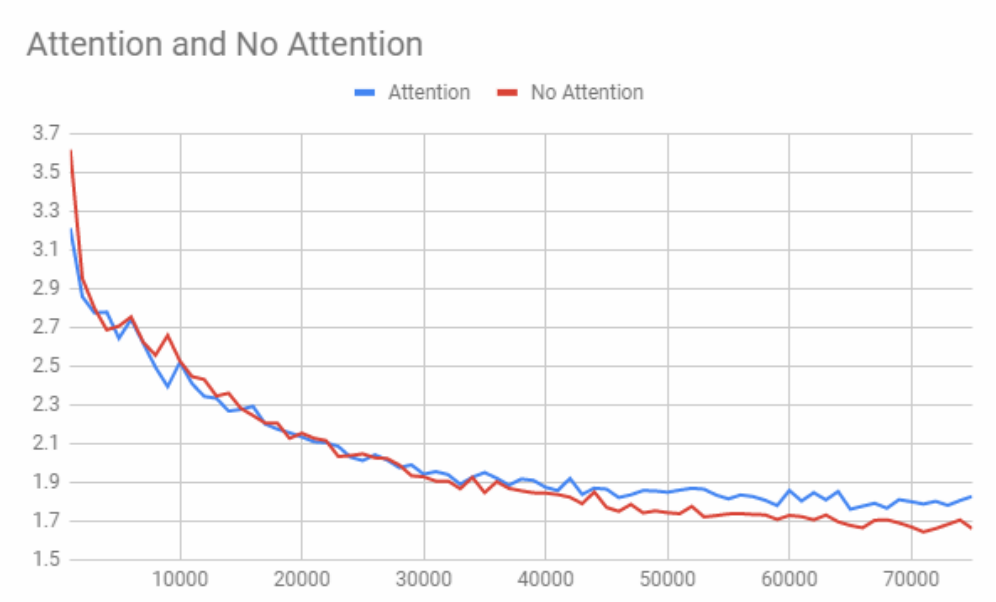


The most accurate model is RNN_i, which takes the hidden state, previous character, and category at every time step. Providing the category at every time step helps improve the model's predictions because some languages use certain letters more than others. Providing the previous character at every time step allows the model to discover trends in pairs of consecutive letters and use these trends to make more accurate predictions. Since RNN_i has both the previous letter and category as input, it can discover trends in pairs of letters for each language.

The second most accurate model takes the hidden state and previous letter as input, but only takes the category on the first time step. Providing the category in the first iteration makes an imprint on the hidden state; however, subsequent iterations will slowly cause this imprint to fade. Therefore, it is slightly harder for RNN_ii to make predictions based on category at the end of the string.

The third most accurate model only takes the category and hidden state as input. Therefore, this model is making predictions based only on trends between letter frequency and language. Finally, RNN_iv is only passed the category at the first time step and from then on only makes predictions based on the hidden state. Therefore RNN_iv is also making predictions based only on trends between letter frequency and language, but near the end of the string it starts to forget what language its making predictions for. However, the category will probably not fade from the hidden state very quickly because there is no other input to distort the hidden state vector.

1c)



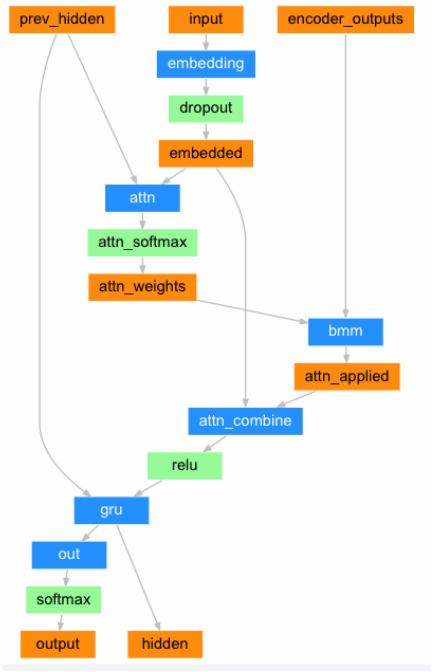
Question 1c uses a sequence to sequence networks to translate sentences from French to English. The encoder of the network is an RNN that condenses an input sentence into a vector. The encoder is passed one word at a time and outputs a vector and a hidden state using a GRU layer. The hidden state is passed back into the encoder to produce the encoding for the next word. The last output vector of the encoder is called the context vector because it encodes the context of the entire sentence.

The decoder is an RNN that uses the encoder's output to produce a sequence of words in another language. Question 1c compares two types of decoders. The simplest decoder uses only the context vector from the encoder to produce its translation. The attention decoder is passed all of the encoder's outputs at every iteration, along with the previous hidden state and previous output. At every iteration the attention decoder uses the previous prediction and hidden state to produce a set of attention weights. The attention weights are multiplied by the encoder outputs to produce a weighted combination. The goal of the weighted combination is to target parts of the sentence that are most relevant. After the attention weights are applied, the decoder produces a prediction for the next word, and a new hidden state.

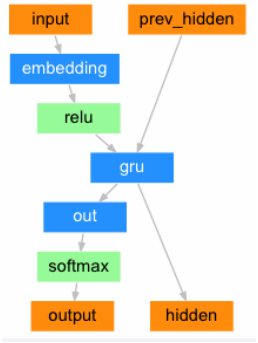
The test results from both versions of the network (with and without attention) are very similar. Both networks learn at a similar rate, but after 75,000 iterations, the network without attention is slightly more accurate (lower negative log likelihood). There are a couple of reasons for this; firstly, the maximum length for any sentence in the given dataset is quite short, only ten words. Keeping the sentence length short makes it easier for the network without attention to make predictions because there is less information that needs to be condensed into the context vector. If the dataset consisted of longer sentences, it would be more difficult for the simple decoder to extract the entire meaning of the sentence from just the context vector. The attention decoder would likely be more accurate at decoding longer sentences because it is given all of the encoder output at every iteration. This, combined with its attention mechanism, allows the attention decoder to focus on the most relevant parts of the sentence when predicting each word.

The second reason that the simple decoder performed slightly better than the attention decoder is that it performs fewer tensor operations. The input to the attention decoder is much larger, and its accuracy requires it to optimize its attention weights as well as the GRU. This added complexity means that the attention decoder takes slightly longer to train.

Attention decoder RNN



Simple Decoder RNN

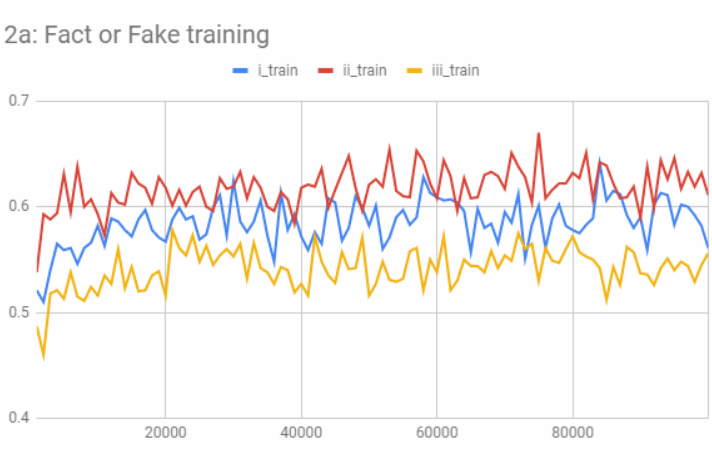


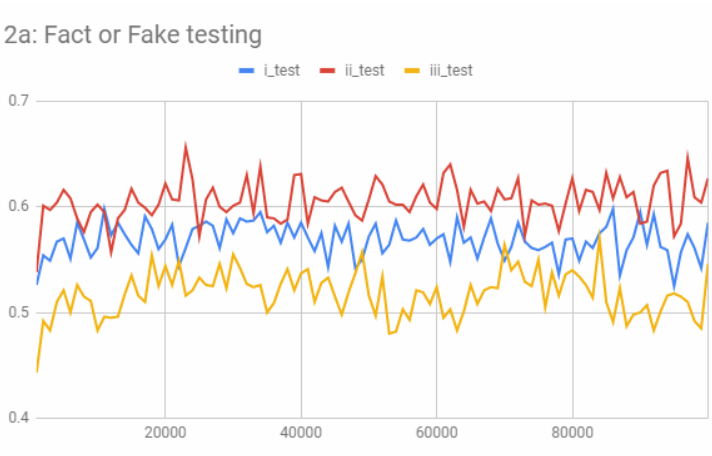
2a)

i_train/i_test: Only the claim

ii_train/ii_test: Claim and claimant are concatenated before being embedded and input to RNN

iii_train/iii_test: Claim is concatenated with claimant and 5 most similar sentences from related articles.





The RNN provided for question 2a, which consists of a single Linear layer, produces context vectors to summarize input sentences. To do this, it is passed tensors of size 50 one at a time which represent sub-words from the input claim. The issue is that the output from the RNN will be affected more from the sub-words at the end of the list then those at the beginning because it does not use any form of memory such as an LSTM or GRU. This can distort the context of the claim and make it harder for the classifier to make an accurate prediction.

For question 2a, the RNN/Classifier is trained to assess the truthfulness of an online claim. There are three possible categories to classify each claim (true, partially true, false), therefore the accuracy of a random guess would be close to 0.33. After 1000 iterations, all versions of the model have average accuracies close to 0.5 which indicates a steep increase in accuracy during the first 1000 iterations. However, this sharp increase in accuracy levels out quickly, and the models do not improve significantly after 15,000 iterations.

The most accurate model produces classifications based on the claim and the claimant. The model that produces classifications based only on the claim is slightly less accurate. This suggests that some claimants tend to make claims of one category more than the others. The model is able to use these claimant trends to predict the validity of each claim more accurately. Moreover, since the claimant information is added to the end of the input, it is not as affected by the RNN's lack of memory.

Finally, the model with the lowest accuracy takes the claim, claimant, and the 5 most similar sentences from related articles as input. This model struggled to make accurate predictions because its input strings are much longer than the input strings of the other two models. It therefore suffers more from the memory issues than the other models. Having more words also makes it more difficult for the model to summarize the entire string in a single context vector.

Sources:

1. Robertson, Sean. "Generating Names with a Character-Level RNN¶." *Generating Names with a Character-Level RNN - PyTorch Tutorials 1.1.0 Documentation*, 2017, pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html.

2. Robertson, Sean. "Translation with a Sequence to Sequence Network and Attention¶." *Translation with a Sequence to Sequence Network and Attention - PyTorch Tutorials 1.1.0 Documentation*, 2017, pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.