**CS 4740 Project 3 (Part 2) Write Up Document**

**Section 1 Questions**

**→ Q1.1: Representation**

Each x\_i needs to be produced in some way and should correspond to word i in the text. This is different from the text classification approaches we have studied previously (BoW for example) where the entire document is represented with a single vector. Where and how is this being done for the RNN?

**Your answer:** We transform each word into a word embedding vector using the glove-wiki-gigaword-300 from gensim.downloader, as suggested in the code comments. This produces a 300 long vector, which we then feed into our RNN. For unknown words in the word embeddings, we feed in an embedding vector of just zeros. This is being done early in the forward function of the

**→ Q1.2: Initialization**

There will be weights that you update in training the RNN. Where and how are these initialized?

**Your answer:** We use pytorch’s default randomized weight initialization to initialize the weights our models use. The linear layer’s weights and biases (if and where applicable) are generated from a uniform random distribution from -sqrt(k) to sqrt(k), where k is 1 divided by the length of the input layer. Similarly, the RNN’s weights and biases for each layer (if and where applicable) are generated from a uniform random distribution from -sqrt(k) to sqrt(k), where k is 1 divided by the length of the corresponding layer.

**→ Q1.3: Training**

You are given the entire training set of N examples. How do you make use of this training set? How does the model modify its weights in training (this likely entails somewhere where gradients are computed and somewhere else where these gradients are used to update the model)? Note: This is code you may not have written but that we have written for you!

**Your answer:** We made use of the training set by creating a set of pre-trained embeddings and passing them into our model

**→ Q1.4: Model**

This is the core model code, ie. where and how you apply the RNN to the x\_i?

**Your answer:** To run the model on input to actually get an output, we use the function NMT\_model.beam\_search(input). This function performs a beam search and attempts to determine which output is the most likely given the input, using the decoder.

**→ Q1.5: Stopping**

How does your training procedure terminate? Note: This is code you may not have written but that we have written for you!

**Your answer:** Once the model has looped through the set amount of epochs, defaulted at 10, in the Train\_and\_evaluate function, the training process terminates.

**→ Q1.6: Hyperparameters**

To run your model, you must fix some hyperparameters, such as $h$ (the hidden dimensionality of the z\_i referenced in the instructions). Be sure to exhaustively describe these hyperparameters and why you set them as you did ( this almost certainly will require some brief exploration: we suggest the course text by Yoav Goldberg as well as possibly the PyTorch official documentation). Be sure to accurately cite either source.

**Your answer:**

Embed Size: our embed size is 300 because that is the size of the pretrained embeddings that we feed into the Model

Hidden Size: our hidden size is 256 because we noticed that a hidden layer of 512 is too larger and makes the BLEU score erradic and doesn't stably increase.

Epochs: We set these at 10 because we mostly because of computational power and time restriction

Train Batch Size: we set our training batch size to 32 because it performs better than with a size of 16. The training run time is faster and we get more accurate results

Clip Grad: we kept this at 2 because our neural network isn't that deep.

Pretained\_source: we set the the pretrained source embeddings with our embeddings that we made from GLoVe300. Giving it pretrained embeddings helps the model perform better than not having any

Pretrained\_target: we set that to none because we had no pretrained embeddings that it could input

**Section 2 Questions**

**→ Q5: Ablation study**

In Part 2.1: Within-Model Comparison, you will need to study what happens when you change parameters within a model

A large aspect of rigorous experimentation in NLP (and other domains) is the ablation study. In this, we ablate or remove aspects of a more complex model, making it less complex, to evaluate whether each aspect was neccessary. To be concrete, for this part, you should train 4 variants of the RNN model and describe them as we do below:

1. Baseline model
2. Baseline model made more complex by modification 𝐴 (e.g. changing the hidden dimensionality from ℎ to 2ℎ ).
3. Baseline model made more complex by modification 𝐵 (where 𝐵 is an entirely distinct/different update from 𝐴 ).
4. Baseline model with both modifications 𝐴 and 𝐵 applied.

Under the framing of an ablation study, you would describe this as beginning with model 4 and then ablating (i.e. removing) each of the two modifications, in turn; and then removing both to see if they were genuinely necessary for the performance you observe.

Once you describe each of the four models, report the quantitative bleu score/ perplexity. Conclude by performing a nuanced analysis.

The descriptive analysis can take one of two forms:

1. **Nuanced quantitative analysis:**
   1. If you choose this option, you will need to further break down the quantitative statistics you reported initially. We provide some initial strategies to prime you for what you should think about in doing this: one possible starting point is to consider: if model 𝑋 achieves greater accuracy than model 𝑌 , to what extent is 𝑋 getting everything correct that 𝑌 gets correct? Alternatively, how is model performance affected if you measure performance on a specific strata/subset of the source sentences?
2. **Nuanced qualitative analysis** 
   1. If you choose this option, you will need to select individual examples and try to explain or reason about why one model may be getting them right whereas the other isn’t. Are there any examples that all 4 models get right or wrong and, if so, can you hypothesize a reason why this occurs?

**Your answer:**

For our ablation study, we train four models and vary

A: the number of hidden layers in the RNN

B: the learning rate of the model.

All models have:

Hidden layer size = 256,

Embeddings size = 300, using embeddings from glove-wiki-gigaword-300

Epochs = 10

Train batch size = 32

We simplify our most complex model in A and B to study which of A and B are necessary to improve the model.

Model 4, is our most complex model and has (hidden layers = 2, learning rate = 5e-4)

Model 3 simplifies A and leaves B (hidden layers = 1, learning rate = 5e-4)

Model 2 simplifies B and leaves A (hidden layers = 2, learning rate = 1e-3)

Model 1 simplifies both A and B (hidden layers = 1, learning rate = 1e-3)

We decided to modify the number of hidden layers because, in class, when discussing RNN’s, we discussed that increasing the number of hidden layers allows the model to represent higher-level information in the RNN, and to allow previous words to potentially have a more complicated impact on the output of the RNN.

We decided to modify the learning rate because we were aware that if our learning rate is too high, it can cause the model to overfit and therefore miss the minima that it is aiming for. However, if the learning rate is too low, the model can take a lot longer to reach the minima it is aiming for. By varying the learning rate we can estimate whether our choice of learning rate was useful or not.

Reports from the models:

| Model number | Included | Hidden layers | Learning rate | Bleu Score | Validation perplexity |
| --- | --- | --- | --- | --- | --- |
| 4 | A and B | 2 | 5e-4 | 3.5 | 39.16 |
| 3 | B | 1 | 5e-4 | 3.72 | 39.304 |
| 2 | A | 2 | 1e-3 | 2.49 | 38.00 |
| 1 | neither | 1 | 1e-3 | 1.78 | 37.80 |

For our nuanced analysis, we attempt to perform qualitative analysis on a group of 5 input out pairs, generated from each model.

| Model 4 (A and B) | Learning Rate 5e-4 and 2 hidden layers |
| --- | --- |
| Input | Output |
| Why should i play the roman fool and die on my own sword | Why have you been gadding? |
| How was your day | She was false. |
| My mind is an Enigma | Tis strange. |
| So it seems there may be some confusion regarding the initialization component of the writeup | I’ll have no more marriages. |
| I’m launching a rocketship to mars | It is. |

| Model 3 (B) | Learning Rate 5e-4: |
| --- | --- |
| Input | Output |
| Why should i play the roman fool and die on my own sword | What would i see the issue of the poisoning? |
| How was your day | so. |
| My mind is an Enigma | Alack the day? |
| So it seems there may be some confusion regarding the initialization component of the writeup | Those that hobgoblin are but sheep |
| I’m launching a rocketship to mars | By heaven art thou finished! |

| Model 2 (A) | 2 hidden layer model |
| --- | --- |
| Input | Output |
| Why should i play the roman fool and die on my own sword | Why should i bear for this |
| How was your day | roderigo! |
| My mind is an Enigma | Who comes here! |
| So it seems there may be some confusion regarding the initialization component of the writeup | Indeed they are disproportion’d |
| I’m launching a rocketship to mars | indeed! |

| Model 1 (neither) | baseline |
| --- | --- |
| Input | Output |
| Why should i play the roman fool and die on my own sword | why should she be angry? |
| How was your day | nothing else. |
| My mind is an Enigma | popilius lena is marr'd. |
| So it seems there may be some confusion regarding the initialization component of the writeup | haply the brinded cat hath mew'd. |
| I’m launching a rocketship to mars | hood mine eyes are made of grapes. |

Frankly, the outputs of our models leave a lot to be desired. Most of our output sentences don’t tie in well with our input sentences at much. We believe that the content from our input sentences is not being properly conserved when the output sentences are generated. This is supported by the fact that our bleu scores are quite low. Even though our models are not performing as well as they could, we can still perform an interesting qualitative analysis of the outputs the model is returning.

Firstly, note the example provided in the code, ‘why should i play the roman fool and die on my own sword’. All of the models we use in this ablation study successfully return outputs that are phrased as questions. This indicates that the model is able to preserve some content across, and is at least able to represent the fact that the input is a question.

Furthermore, the outputs of our models aren’t completely nonsensical. Most of the outputs are grammatically correct, and many of them convey some real meaning. Sentences like ‘why should she be angry’, and ‘ill have no more marriages’ are strange and often do not share context with the input sentences, but they are grammatically correct and convey some sensible and nontrivial meaning.

Our models also seem to be using a lot of basic general phrases to represent the inputs. Several of our input sentences are converted to simple phrases such as ‘indeed!’, ‘roderigo!’, ‘so’, and ‘it is’. Which all of our models produce such phrases, our model 2 seems to produce these phrases more often. Perhaps this indicates that our models are simply not making enough use of the inputs and are defaulting to using basic inputs.

Model 1 and 2 also seem to use proper names such as ‘popilius lena’ and ‘roderigo’ for some of the translations. Since the inputs don’t include proper names, this is a strange and major flaw. Model 3 and 4 seem to not use proper names. Since models 3 and 4 have higher Bleu scores, as well, this suggests that they are better at conserving the inputs and converting them into a reasonable response, and therefore that the lower learning rate might be a more key factor in getting better results.

**Section 3 Questions**

**→ Q3.1:**

Earlier in the course, we studied models that make use of Markov assumptions. Recurrent neural networks do not make any such assumption. That said, RNNs are known to struggle with long-distance dependencies. What is a fundamental reason for why this is the case?

**Your answer:** Because when we are calculating the loss during backpropagation, we are repeatedly calculating dots products and this can lead to a vanishing gradient for long-distance dependencies. The gradient signal from far away is lost because the gradient signal from nearby is much greater. It's smaller because while computing the dot products, it is doing so with numbers that are < 1.

**→ Q3.2:**

In applying RNNs to tasks in NLP, we have discovered that (at least for tasks in English) feeding a sentence into an RNN backwards (i.e. inputting the sequence of vectors corresponding to ( 𝑐𝑜𝑢𝑟𝑠𝑒 , 𝑔𝑟𝑒𝑎𝑡 , 𝑎 , 𝑖𝑠 , 𝑁𝐿𝑃 ) instead of ( 𝑁𝐿𝑃 , 𝑖𝑠 , 𝑎 , 𝑔𝑟𝑒𝑎𝑡 , 𝑐𝑜𝑢𝑟𝑠𝑒 )) tends to improve performance. Why might this be the case?

**Your answer:** Feeding in the sentence backward allows the RNN to build up the context for the subject word. In the example, the backward RNN is able to use words like ‘great’ and ‘course’ to get an idea of what the subject (in this case ‘RNN’) means and how it should be handled. Since an RNN makes decisions on the output of a word before looking at the next word, a forward RNN would not be able to make use of the context and would have to process ‘RNN’ without any context. Since ‘RNN’ is unlikely to have an embedding in common embedding databases, this means that the model is unlikely to be successful.

Also, in an RNN, as discussed in class, each word is passed through a neural network, and the outputs of the hidden layers in that network are passed into the hidden layers of the next word’s network as inputs.Since the output generated by an RNN is the output of the neural network when it is run on the last word, this can lead to the last word having a greater impact on the output of the RNN than words processed earlier. This might be a factor that can cause improvement in the performance of the RNN when feeding the input vector in backward, perhaps because the first words in an English sentence are more likely to hold key information than the last words.

**→ Q3.3:**

In using RNNs and word embeddings for NLP tasks, we are no longer required to engineer specific features that are useful for the task; the model discovers them automatically. Stated differently, it seems that neural models tend to discover better features than human researchers can directly specify. This comes at the cost of systems having to consume tremendous amounts of data to learn these kinds of patterns from the data. Beyond concerns of dataset size (and the computational resources required to process and train using this data as well as the further environmental harm that results from this process), why might we disfavor RNN models?

**Your answer:** Because of the complex patterns that the RNN detects, it is much more challenging to see where our neural model is going wrong since we can't easily point to a certain section of our code. There are a lot more logical errors than runtime errors when working with Neural Networks. Also, RNN's (Without attention) have a vanishing gradient, meaning that the context

Because of the need for an enormous training dataset, we have to process all of that data which in turn makes training the model significantly long.

**Section 4Questions**

**→ Q4: Miscellaneous**

List any additional libraries you used and sources you referenced and cited (labelled with the section in which you referred to them). Include a description of how your group split up the work. Include brief feedback on this assignment.

**Your answer:** I feel like there wasn't enough time given to work on this project. We worked on this. To get any real progress in this assignment while training the model, we had to buy Google colab pro. We have no clue how others would be training their models without the pro version of google colab considering it would take several hours to train a model. We probably spent a total of about 22 hours working on this project and we completed it together using pair programming.