DS-GA 1011 Final Project: Neural Machine Translation

Kimberly Hafner `kfh226`

Jingshu Liu `jl7722`

Taurean Parker `tp823`

Alexandra Simonoff `ams889`

1 Introduction

For our final project, we built three different encoder-decoder models: a Recurrent Neural Network (RNN) without attention, an RNN with attention and an RNN with a Self-Attention based encoder. We ran models using provided pre-processed training datasets for Vietnamese to English translation and Chinese to English translation, and during hyperparameter tuning, we used the corpus BLEU score on the validation data to gauge generalized performance. We then ran the best model for each language and model type pair and reported on test corpus BLEU score.

Full analysis and code can be found on the Github Repo

2 Models and Hyperparameter Tuning

In this section, we discuss the three types of model explored. If not otherwise specified, all models are trained with Adam optimizer and exponentially decaying learning rate every 10 epochs by a ratio of 0.9. We used minibatch size 64 and each epoch typically contains 1000 batches. During training, if teacher forcing is used, the correct target from previous step is used as input to the decoder; otherwise the word with the highest predicted probability from previous step is used as input. We used maximum sentence length 50 for both input and target sequence. The maximum vocabulary size of both input and target language is set to 100K for the RNN models, and 25K for the other models. Further discussion can be found in Section 2.4.3.

2.1 RNN Based Encoder-decoder without Attention

In this subsection we show the results using a recurrent neural network encoder-decoder model without an attention mechanism.

2.1.1 Vietnamese to English

For the Vietnamese to English model, we decided to tune several different hyperparameters to gauge the overall effect and inform the general tuning baselines moving forward with the Chinese and English model. In order to test as many options as possible while staying within our budget, we used a batch size of 32 and up to 100 epochs containing 200 batches.

Learning Rate
First we looked into learning rate and tested rates between 1e-2 and 1e-4. A learning rate of 0.001 appears to perform best for our model; Smaller learning rates appear to anneal too fast, larger learning rates appear volatile and and result in low generalized performance. At 0.001 our BLEU tops 9.3 after 100 epochs and training loss drops below 55. We will move forward using a learning rate of 0.001.

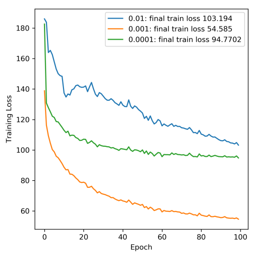


Figure 1: Learning Rate Training Loss

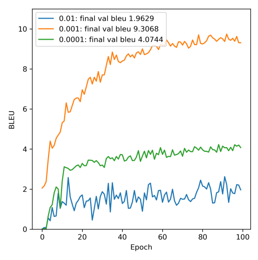


Figure 2: Learning Rate BLEU Scores

Drop Out Rate
Next, we dove into drop out rate testing 0, 0.3 and

0.6 drop out rates.

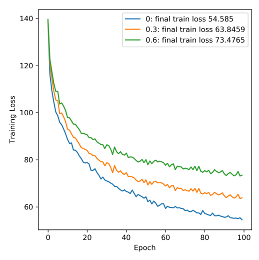


Figure 3: Drop Out Rate Training Loss

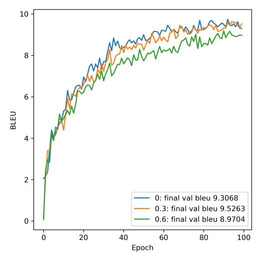


Figure 4: Drop Out Rate BLEU Scores

Drop out rates of 0 and 0.3 perform nearly identically on our validation set with drop out rate 0 having a higher overall average across the last 50 epochs. The training loss is also lower for drop out 0 as expected but it does appear to generalize well when looking at the nearly identical BLEU scores of 9.3 and 9.5 for drop out rates 0 and 0.3 respectively. Given the higher average BLEU we will move forward with a drop out rate of 0.

Hidden Dimension Size

Next, we experimented with a wide range of hidden dimension sizes between 128 and 1024.

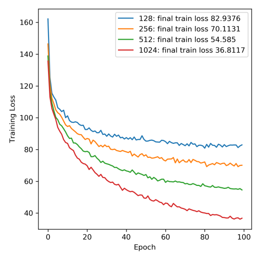


Figure 5: Hidden Size Training Loss

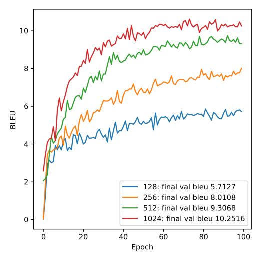


Figure 6: Hidden Size BLEU Scores

As hidden dimension size increases, BLEU score increases and training loss decreases. It appears that the improvement of doubling hidden dimension size is diminishing as we increase past 512 given the gain 1024 has over 512 is much less profound than the gain 512 has over 256. We will not double hidden size again to 2048 as that will likely not improve BLEU very much as it begins to overfit our training set. We will use a 1024 hidden size for our best model (though we will use 256 or 512 in tuning additional parameters as the 1024 model takes significantly longer to run and the results would likely be very similar with respect to rankings of hyperparameters).

Pretrained vs Non Pretrained Embeddings

Given we were running many models and wanted to get a sense of performance within a time window, we did all of the previous calculations using pretrained embeddings. For the sake of building the best model we possible could, we also tested not using a pretrained embedding and instead letting the model develop the embedding that best fit our dataset.

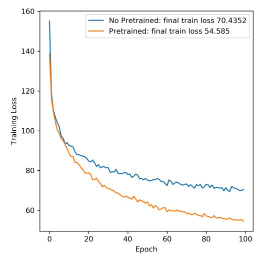


Figure 7: Embedding Style Training Loss

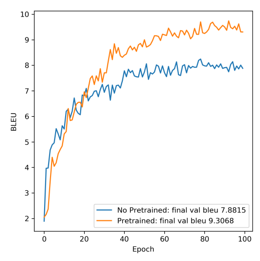


Figure 8: Embedding Style BLEU Scores

While the non pre-trained embedding appears to perform better towards the earlier stages of model training, ultimately our pretrained embedding-based model does significantly better and also takes half the time to run.

Other Parameters Tested

In addition to the parameters mention above we tested several other parameters in an attempt to improve performance. We tested using pretrained embeddings but allowing the model to update these values as a way to create a hybrid between pretrained embeddings and embeddings learned by the model. This resulted in performance slightly worse than our pretrained embeddings model and double the time for computation. We also testing using a bidirectional LSTM in place of our encoder but the results didn't improve on our model with the same encoder hidden dimension size. Lastly, we tested using teacher forcing ratios .2 and 1 and found that a teacher forcing ratio of 1 gave us the best results. Therefore we found our best model would use a learning rate of 0.001, a drop out rate of 0, a hidden dimension size of 1024, and teacher forcing of 1.

2.1.2 Chinese to English

We explored the model using both pre-trained and non-trained word embeddings with a initial hidden dimension size of 256 and teaching force ratio less than 1. However, after noticing poor results on our BLEU score metric, we increased the hidden dimension size to 512. The following plots below are based on the aforementioned information:

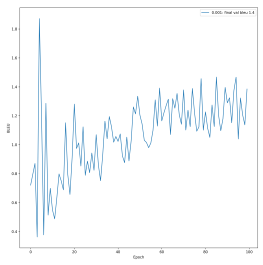


Figure 9: Learning Rate BLEU Scores(Non-Pretrain Vectors)

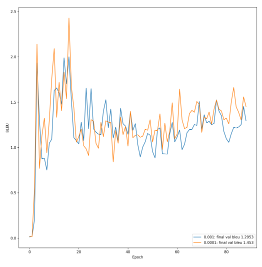


Figure 10: Learning Rate BLEU Scores(Pretrain Vectors) for Learning Rate 0.001 & .0001

Learning Rate

Before exploring different learning rates on the Chinese- English translation model, we chose an initial rate learning rate of .001 to compare model performance between the pre-trained word embeddings and the non-trained word embeddings. Overall, when using a learning rate .001, we observed that final BLEU score of 1.29 for pre-trained vectors and 1.40 for non- trained vectors. In additional to exploring model performance for both pre-train and non-train vectors, we reran the model using a lower learning rate .0001 on the pre-trained vectors and observed a BLEU score of 1.45. This is a significant improvement from using a higher learning rate .001, and thus concluded that .0001 is a suitable learning rate. In both cases, the model is learning a slowly, but we achieved better results in a shorter amount of time.

Drop Out & Teaching Force

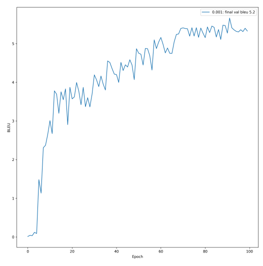


Figure 11: BLEU Score(Pretrain Vectors) & Teaching Forcing

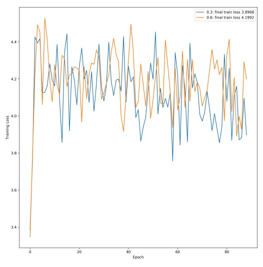


Figure 12: Training Loss by BLEU Scores(Pretrain Vectors) for Dropout Rate .03 & .06

Next, we tried different dropout rates for the Chinese-English translator. We ran the model with a dropout 0.3 and 0.6. In both cases, the model performed unfavorable as the training loss and validation loss increased significantly. By observing from the figure, we noticed that while the training loss for the dropout 0.3 is lower than 0.6, but the BLEU score for the dropout rate 0.6 is above 4 and the 0.3 dropout rate is hovers around 3.8. Based on these observations, we determined that that a dropout of 0 would be more suitable, as increasing the dropout led to degraded model performance despite having a higher BLEU score. After tuning the dropout and learning rates, we decided to change the teaching force ratio to 1, and notice significant improvement in our BLEU score, which a observed a score above 5. In addition, we had lower scores on training and validation loss in comparison when the teaching force ratio was less than 0.5.

2.2 RNN Based Encoder-decoder with Attention

In this section we discuss parameter tuning on RNN with the attention mechanism. The encoder is a bi-directional GRU model as described in Section 2.1. The decoder is a GRU model with attention on the encoder hidden layers across all time steps. Decoder hidden states are initiated by the encoder hidden states, concatenating the last step from both the forward and backward pass in the bi-directional GRU.

To calculate the attention weights, we use the dot product between a query vector and the encoder hidden layer vector corresponding to each input word. We normalize the weights by the softmax function, and calculate the weighted-average of the encoder hidden layers as the context vector. The decoder then takes both the context vector and the embedding from previous step prediction as input to the current step. The query vector is constructed by concatenating the embedding of the decoder output and decoder hidden state from the previous step, and then projected to the same vector space as the encoder embedding vectors.

For attention models, we mainly explored different learning rates, hidden dimensions of the encoder and drop out rates. The hidden dimension of the decoder GRU is twice the size of the encoder one. Teacher forcing ratio is set to 1 based on early exploration.

2.2.1 Vietnamese to English

Learning Rate and Hidden Dimension As shown in Figure 13, we tested combinations between learning rate 1e-3, 1e-4, and encoder hidden dimension 256, 512 and 1024. Dropout probability is set to 0.5, and models are trained up to 100 epochs. An initial learning rate of 1e-3 and dimension 256 gives superior performance early on, but seems to quickly overfit. Instead, a learning rate of 1e-4 and dimension 512 shows the best result towards the end, achieving a BLEU score of 13.8 on the validation set.

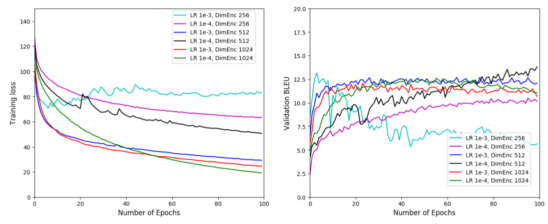
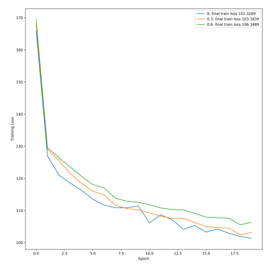


Figure 13: Learning rate and encoder hidden layer dimensions

Drop Out

As shown in Figure 14 and Figure 15 we tested dropout probabilities of 0, 0.3, and 0.6 after tuning the learning rate and hidden dimension using dropout probability of 0.5. This was trained on 20 epochs so the final BLEU score was low, but it is apparent from these results that the dropout of 0 performed best, giving a BLEU score of 4.0307 and the lowest training loss of 101.3269.



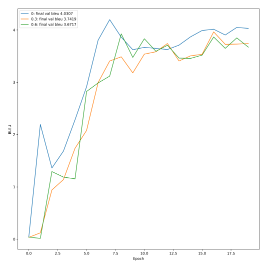


Figure 14: Drop Out Rate Training Loss

Figure 15: Drop Out Rate BLEU Scores

2.2.2 Chinese to English

Similar to the Vietnamese model, we tested combinations of learning rate (1e-3, 1e-4) and hidden dimensions (256, 512, 1024). As shown in Figure 16, the validation performance are similar though the BLEU score is relatively lower. Additionally, we tried to update the pretrained embedding matrices. Unlike in the vanilla RNN model, updating pretrained embedding shows superior performance (green line in figure). The validation BLEU is around 11.

Given time limit, we did not finish training the Vietnamese to English model that updates the embeddings.

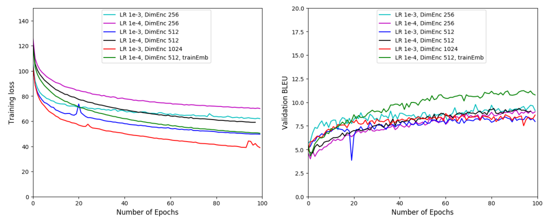


Figure 16: Learning rate and encoder hidden layer dimensions

Drop Out Again as in the Vietnamese model, we tested the dropout probabilities of 0, 0.3, and 0.6. The resulting training loss and BLEU scores are shown in Figure 17 and Figure 18 and the models were trained over ten epochs. For the Chinese corpus, the dropout of 0 still had the lowest training loss, 108.9537, but the dropout of 0.6 actually produced the highest BLEU score, 3.4788, and the highest training loss, 113.0557.

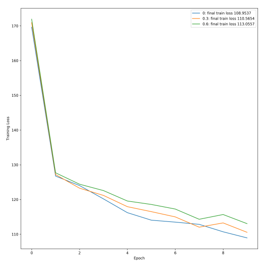


Figure 17: Drop Out Rate Training Loss

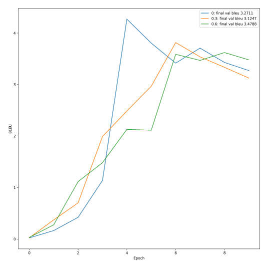


Figure 18: Drop Out Rate BLEU Scores

2.3 Self-attention Based Encoder with RNN Based Decoder

We implemented the encoder with self-attention following (?). We used the same type of decoder as described in Section 2.2 and attended to the output from the last stack of the encoder. In addition, we used the (projected) average of the encoder output to initiate the decoder hidden states.

For the self-attention model, we explored a various range of hyper-parameter settings, including: initial learning rate (1e-2, 1e-3, 1e-4, 1e-5), decoder hidden layer dimension (512, 1024), encoder feed-forward layer dimension (512, 1024), number of stacks in encoder (2, 4, 6, 8, 10), teacher forcing ratio (1, 0.5) and fix/update pretrained embeddings. However, none of the settings led to similar performance as other models. The best validation BLEU score is around 8 after 40 epochs, while most of the attention based models described in 2.2 go above 10 at epoch 40. Performance curves can be found in the appendix.

2.4 Miscellaneous

In addition to parameters described in above sections, we also explored the following items. Most of the alternatives do not make a substantial difference.

2.4.1 Loss calculation

When not using teacher forcing, we mask the loss after the prediction generates the EOS token. We then tried 1) averaging the loss across all non-masked elements in the entire batch, and 2) sum within the sample and average by batch size. We did not find significant difference in result. Model performance reported in this report used the second approach.

2.4.2 Padding tokens

We noticed from the attention visualizations such as Figure 20 that positive weights can be assigned to padding elements. We suspect this might introduce noise, and one way to alleviate it is to use the pack_padded_sequence functionality in pyTorch and only run forward pass (and backward pass) on non-padded elements. We tested it both on the RNN model and the attention based models. Interestingly, with RNN model it seems to lead to superior performance, getting to BLEU around 12 after 100 epochs on the Vietnamese task. Yet we did not observe improvement on the attention based models within 100 epochs. We are limited by time but running the model longer may generate different conclusions.

Additionally, we tested masking the padded elements from encoder output when calculating the attention weights as done in the self-attention models. We did not observe improvements in performance within 100 epochs either.

2.4.3 Vocabulary size

We started with a maximum vocabulary size of 100K with the vanilla RNN models, and later reduced it to 25K with the attention and self-attention models. We did not notice any obvious performance difference between the two options.

2.4.4 Sequence length

We tried both target length of 40 and 50, based on the 90th and 95th percentile of the target sequence. No obvious difference found.

2.4.5 Beam search

The predictions made by the model when using beam search were not much improved upon our results using greedy search. Using beam search would often predict the end of sentence token early in parsing and would hinder the BLEU score with a brevity penalty. Based on these results, and the fact that it took longer to run, we used greedy search in the final model.

3 Results and analysis

3.1 Validation results summary of selected models

The table below summarizes the validation BLEU from several selected models.

Table 1: Validation BLEU of selected models

	Vietnamese	**Chinese**
RNN-1	10.0	N/A
RNN-2	N/A	5.4
Att-1	13.8	9.4
Att-2	N/A	11.2
Self-Att	7.9	N/A

Table 2: Model descriptions

RNN-1	Hidden dim 1024, learning rate 0.001, dropout rate 0, teacher forcing rate 1
RNN-2	Hidden dim 512, learning rate 0.001, dropout rate 0.5, teacher forcing rate 1
Att-1	Encoder dim 512, learning rate 1e-4, dropout rate 0.5
Att-2	Encoder dim 512, update embedding
Self-Att	Decoder dim 1024, feed-forward dim 512, trained up to 40 epochs

3.2 Best model test set performance

Applying the best RNN without attention models, we achieved BLEU scores of 10.9 and 6.7 on the Vietnamese to English task and Chinese to English task, respectively, using the test dataset. For our top attention based models, we achieved BLEU scores of 14.3 and 11.7 on the Vietnamese to English task and Chinese to English task, respectively, using the test dataset.

3.3 Attention analysis

The following two figures are examples of the attention weights based on the Chinese-English task. In the appendix, Figure 19, the model seems to have learned to correctly align 'scared', 'but' and the second 'we' to the correct Chinese word. Yet it failed to align the first 'we' or 'school'. In Figure 20, the model correctly aligns the word 'people' and 'think', but failed on other cases. In general we observe the model still have a hard time aligning the right terms, especially when the sequence is long.

3.4 Conclusion

Overall, the vanilla RNN and the RNN with Attention models both show increased performance over a short span of time. However, the RNN with Attention model returned the highest BLEU for both translating text from Vietnamese to English and Chinese to English, with Vietnamese tranlations outperforming Chinese in all models.

Below are a few areas we could have explored to further improve model performance, if time permits:

- Train longer. Based on the performance plots, most of our model has increasing validation BLEU score and can still be underfitted with the limited number of epochs. Also based on the attention plots, it seems the model just started to learn to align the source and target words. Training the model with more epochs can probably further improve model performance, if we had more time and computation resource.
- Other form of regularization. We used dropout as the primary regularization method. Even though most of our model do not seem to overfit yet, having additional regularization such as adding random noise to embedding vectors may improve the robustness of the model.
- Better pre-processing. We noticed later in the process that the Chinese input dataset may contain more than one white space between words, which might introduce noise. Also building the token by characters instead of the provided tokens may be more suitable for Chinese language.

A Appendices

A.1 Attention plots

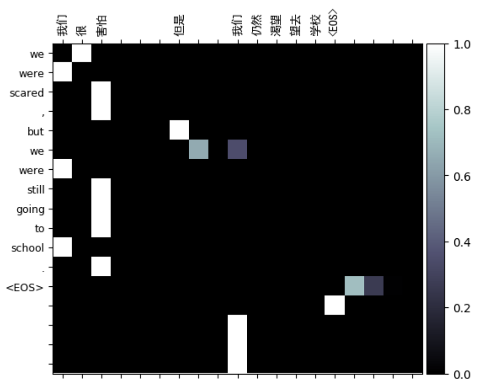


Figure 19: Attention example: Chinese to English A

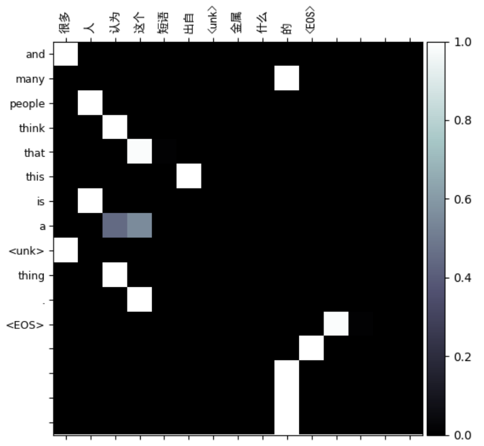


Figure 20: Attention example: Chinese to English B

A.2 Performance figure of self-attention models

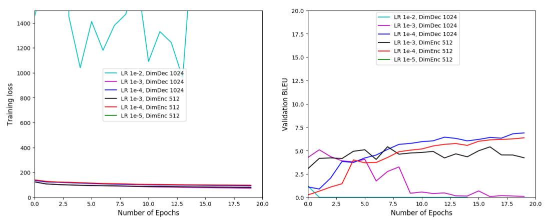


Figure 21: Learning rate and decoder hidden layer dimensions, Vietnamese to English

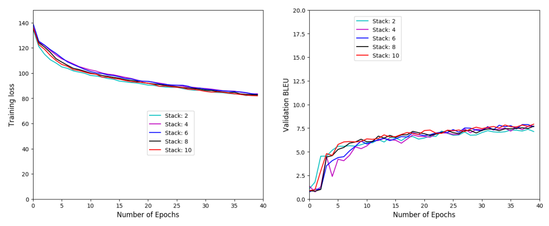


Figure 22: Number of encoder stacks, Vietnamese to English

B Contributions

Below we will outline the work each teammate completed:

Kim: Worked on the RNN with attention model, beam search and evaluation functions.

Jingshu: Worked on the RNN with attention model and self-attention model, turned team notebooks into runnable python files with accompanying shell scripts.

Taurean: Worked on RNN without attention model, data loading and train/train iters functions.

Alex: Worked on gcloud VM setup, RNN without attention model, data loading, train/train iters functions and beam search.