Neural Machine Translation: Akkadian Cuneiform to English

Olivia McCauley | Daniel Bostwick | Nikita Chauhan

Datasci 207 — Spring 2024

1 Introduction

Akkadian cuneiform is a low-resource ancient language, and there has only been a single published paper on neural machine translation in this area using a CNN-based model implemented with Fairseq[3]. The data for this project is published on ORACC, the Open Richly Annotated Cuneiform Corpus, which has been consolidated by the FactGrid Cuneiform project. We propose using several models to establish baseline performance against which other models can be compared, which rely on the foundational papers in modern machine translation by Sutskever[5], Bahdanau[2], and Vaswani[1]. We propose using an LSTM, Seq2Seq LSTM, and a Seq2Seq LSTM with Attention model to establish these baselines, and evaluate model performance with BLEU-4.

2 Objective

Using Neural Machine Translation, we would like to translate Akkadian cuneiform text into English.

2.1 Problem Statement

Thousands of Akkadian texts have already been translated, but there are hundreds of thousands left without translations. We need to create a generalizable model that can accurately translate these Akkadian texts into English to assist historians in understanding our past.

2.2 Approach / Methodology

We approached our objective by training eleven different models that generate English transliterations from Cuneiform inputs. Our design choices ranged from small three layered Recurrent Neural Networks to large scale models with attention. In general, Neural Machine Translation is a very difficult task even with input and output data that share very similar distributions, this might some uniform-like distribution for both sequence length and unique token count. Our data did not possess either of these; however, we will explain more about this in the Dataset section below. Due to these harsh distributions, most of our earlier models, such as our simpler RNN and LSTM models, we narrowed our maximum sequence length to fifty to allow us to capture as much of the data as we could to train on. Luckily for us, models with attention don't mind the unevenness as much and longer inputs can actually improve model performance.

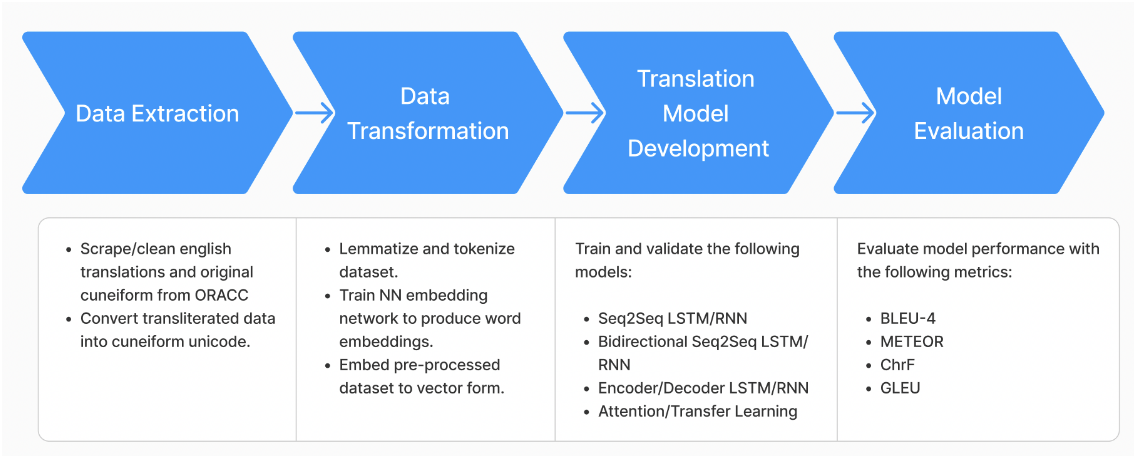
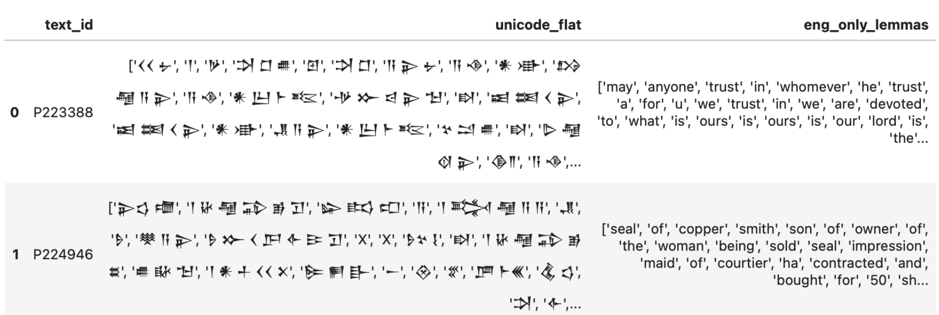


Figure 1: Model Workflow

2.3 Dataset

The dataset we used consisted of three columns: a **Text ID**, a **Unicode List**, and an **English List**. Earlier we mentioned that our data had very interesting distributions in terms of both sequence length and unique token count, this is due to the fact that we also had three different dialects of ancient cuneiform. We believe this is also a result of how our data was structured from the start, our first column, **Text ID**, represents the text in its entirety making our sequence lengths extremely long. In ancient Mesopotamia, Akkadian would be written on clay tablets and we can manually translate them line by line which would probably make machine transliteration significantly easier.



	text_id	unicode_flat	eng_only_lemmas
0	P223388	[illegible]	['may', 'anyone', 'trust', 'in', 'whomever', 'he', 'trust', 'a', 'for', 'u', 'we', 'trust', 'in', 'we', 'are', 'devoted', 'to', 'what', 'is', 'ours', 'is', 'ours', 'is', 'our', 'lord', 'is', 'the'...
1	P224946	[illegible]	['seal', 'of', 'copper', 'smith', 'son', 'of', 'owner', 'of', 'the', 'woman', 'being', 'sold', 'seal', 'impression', 'maid', 'of', 'courtier', 'ha', 'contracted', 'and', 'bought', 'for', '50', 'sh...

Figure 2: First two rows of dataset

This dataset is rather small with only 5,694 rows. Their are 7,102 unique English tokens and 50,660 unique Cuneiform tokens. The longest Cuneiform text is 3,823 characters and the longest English text is 5,511 words.

2.4 Images

To elaborate on the distributions of our data, we noticed that the Cuneiform and English lengths were roughly from the same distribution only that the Cuneiform's distribution was slightly translated to the right meaning that on average the Cuneiform sequence length was larger than the English sequence length.

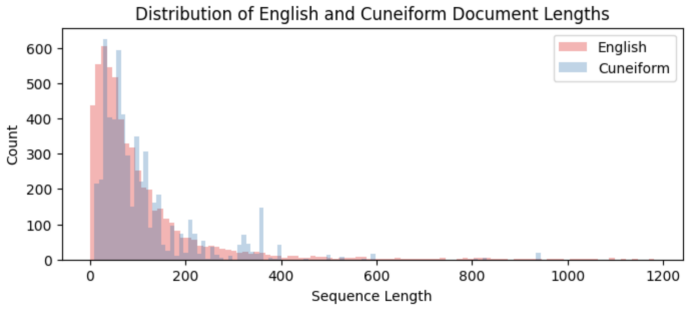


Figure 3: English and Cuneiform have similar distributions overall

2.5 What is considered success / failure?

We propose a two-pronged approach to determining whether our project is a success or failure.

First, the success of our project can be measured against the literature baseline for machine translation on Akkadian Cuneiform, which used BLEU-4 as its sole model evaluation metric. If our models perform on par with the article's model, we consider this project successful.

Second, the success of our project can be measured by meaningful improvements between the baseline and subsequent models, particularly after hyperparameter tuning. If we observe performance gains between architectures, and further improvements after hyperparameter tuning, we consider this project successful.

2.6 Evaluation Metrics

We used the following metrics to evaluate the performance of each model.
First, we used BLEU-4 scoring to evaluate model performance. This corresponds to the amount of time the word shows up in a sentence. BLEU is calculated based on the precision of n-grams (contiguous sequences of n words) in the machine-generated translation compared to the reference translations. The metric penalizes overly short translations and rewards fluent translations that match the references well. We chose BLEU-4 because the original paper translating Akkadian cuneiform to English used four grams to obtain BLEU scores for each sentence pair.

Second, we used METEOR.[6] METEOR is an automatic metric that considers synonyms and paraphrases when evaluating word matches. It computes a harmonic mean of precision and recall, and also includes penalty functions for unigram and stemming. It is a language-independent metric

that correlates well with human judgments of translation quality.

Third, we used ChrF, which is another automatic metric focused on character-level matches rather than evaluating word-level matches like BLEU. It measures the F-score based on the number of character n-grams shared between the machine translation and reference translation. Graphemes in Akkadian cuneiform are character signs that represent syllables, rather than letters that spell out syllables. Because of this, it will be useful to understand how effectively translation occurs on a character level.

The fourth evaluation metric we used was GLEU, which is very similar to BLEU but is Google's version. It works by measuring the distance of the machine translation against the reference translation and counting the number of matching words. It calculates the translation using n-gram precision and is simply the minimum of recall and precision. Similar to BLEU, GLEU penalizes short translations but it also penalizes translations for being too long compared to the references.

3 Experiments

For subsections 3.1 to 3.5 and 3.7, we preprocessed and tokenized our data the same. We first tokenized our input and output data in its entirety, followed by splitting our data into training, validation, and testing sets (60/20/20). We first evaluated these models visually by taking the argmax of each output and detokenizing the predictions. We then detokenized the corresponding known target line and compared predicted and original sequences. We compiled out model with a sparse categorical cross-entropy loss and an Adam optimizer.

3.1 Baseline RNN

This was our rudimentary baseline model that consisted of only three layers. The layers were as follows: first a Gated Recurrent Unit (GRU) with a unit size of 256, second a Dense layer with a unit size of 1024 and a ReLu activation function, third a Dropout layer to regularize our model from within, and lastly another Dense layer with unit size of the output vocab size and an softmax activation function.

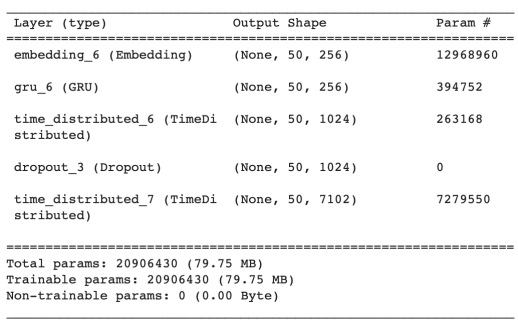
Layer (type)	Output Shape	Param #
gru (GRU)	(None, 50, 256)	198912
time_distributed (TimeDistributed)	(None, 50, 1024)	263168
dropout (Dropout)	(None, 50, 1024)	0
time_distributed_1 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 7741630 (29.53 MB)
Trainable params: 7741630 (29.53 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 4: Baseline RNN Model Summary

3.2 RNN With Embedding Layer

The embedding layer was only one extra thing different from the baseline and that was to add an embedding layer before we introduce the GRU and Dense layers. As you can see from Figure 4, the embedding layer adds a lot of depth to our model adding in over 10 million more parameters to the model. Ideally we would have tuned more



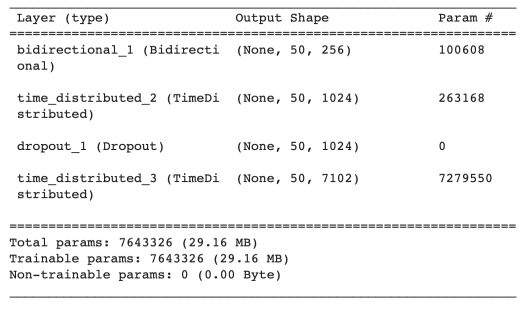
Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	(None, 50, 256)	12968960
gru_6 (GRU)	(None, 50, 256)	394752
time_distributed_6 (TimeDistributed)	(None, 50, 1024)	263168
dropout_3 (Dropout)	(None, 50, 1024)	0
time_distributed_7 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 20906430 (79.75 MB)
Trainable params: 20906430 (79.75 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 5: RNN With Embedding Layer Model Summary

3.3 Bidirectional RNN

The basic bidirectional RNN is almost identical to the baseline RNN in its setup; however, we used a Bidirectional wrapper with the GRU layer to add some complexity to the model. The Bidirectional wrapper generates two separate sets of hidden states, one going forward and one going backward which are then concatenated in a combined representation of the input sequence.



Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirectional)	(None, 50, 256)	100608
time_distributed_2 (TimeDistributed)	(None, 50, 1024)	263168
dropout_1 (Dropout)	(None, 50, 1024)	0
time_distributed_3 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 7643326 (29.16 MB)
Trainable params: 7643326 (29.16 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 6: Bidirectional RNN Model Summary

3.4 Encoder-Decoder RNN

This model implements a different architecture than the last few in that we have two GRU layers, one for the Encoder and one for the Decoder. The Encoder's GRU has the same amount of units as the last models but we also implemented a hyper-parameter that propagates the input data in the reverse direction which is helpful since the order of the input data is very important. We also

added a Repeat Vector layer that specifies the number of times to repeat the input to match the output length. The overall Decoder structure is almost exactly the same as the original models which makes sense since we are trying to output the same as before where the difference is the input of the newly encoded data.

Layer (type)	Output Shape	Param #
gru_6 (GRU)	(None, 256)	198912
repeat_vector_3 (RepeatVector)	(None, 50, 256)	0
gru_7 (GRU)	(None, 50, 256)	394752
time_distributed_6 (TimeDistributed)	(None, 50, 1024)	263168
dropout_3 (Dropout)	(None, 50, 1024)	0
time_distributed_7 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 8136382 (31.04 MB)
Trainable params: 8136382 (31.04 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 7: Encoder - Decoder RNN Model Summary

3.5 Bidirectional Encoder-Decoder RNN

This model is very straight forward given an understanding of the last few models described above. We implemented an Embedding layer before the Bidirectional wrapped GRU to give the model a little more depth. Another difference to the previous model is that both the encoder and decoder have a Bidirectional wrapper on their respective GRU layers.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 50, 128)	6484480
bidirectional_1 (Bidirectional)	(None, 256)	198144
repeat_vector (RepeatVector)	(None, 50, 256)	0
bidirectional_2 (Bidirectional)	(None, 50, 256)	296448
time_distributed (TimeDistributed)	(None, 50, 512)	131584
dropout (Dropout)	(None, 50, 512)	0
time_distributed_1 (TimeDistributed)	(None, 50, 7102)	3643326

Total params: 10753982 (41.02 MB)
Trainable params: 10753982 (41.02 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 8: Bidirectional Encoder - Decoder RNN Model Summary

3.6 Bidirectional Encoder-Decoder RNN With Attention

In this model, we preprocessed our data by tokenizing it as TensorFlow tensor objects instead of vectorizing our data as numpy arrays and tokenizing our inputs and outputs using the keras library functions. Our encoder consists of an Embedding layer and a GRU layer with a Bidirectional wrapper that sums the outputs of the forward and backward sequences. By default, the GRU's weights are initialized as a matrix of orthonormal vectors to stabilize gradients during backpropagation; however, we initialized our weights using random values drawn from a uniform distribution, $[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}]$[4], where n is the number of units in the layer. Our decoder consists of an Embedding layer, a GRU layer without the Bidirectional wrapper, a Cross Attention layer, and a final Dense layer. The attention layer consists of a Multi Head Attention layer, a layer to normalize the activations of the previous layer, and a layer that can add the outputs of different layers as in our case we are adding our RNN output with our attention output.

Layer (type)	Output Shape	Param #
encoder_1 (Encoder)	multiple	407850
decoder_1 (Decoder)	multiple	1731800

Total params: 2139650 (8.16 MB)
Trainable params: 2139650 (8.16 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 9: Bidirectional Encoder - Decoder RNN With Attention Model Summary

3.7 Baseline LSTM

The Baseline LSTM model is still considered one of our baseline models that consisted of only three layers. The layers were as follows: an LSTM layer with a unit size of 512, a Time Distributed Dense layer with a unit size of 1024 and a ReLu activation function, and lastly another Time Distributed Dense layer with unit size of the output vocab size and an softmax activation function. Although LSTMs are generally better than RNNs for learning more in a sequence throughout time, our input data was so long and sometimes sparse that this baseline wouldn't provide many tangible results.

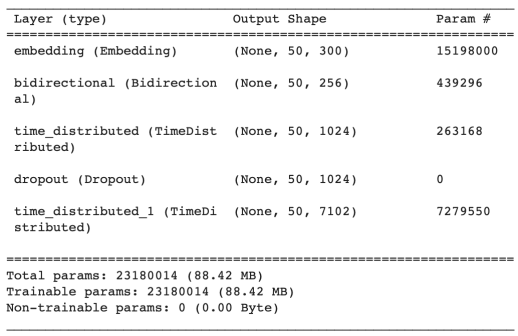
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 512)	1052672
repeat_vector (RepeatVector)	(None, 50, 512)	0
time_distributed (TimeDistributed)	(None, 50, 1024)	525312
dropout (Dropout)	(None, 50, 1024)	0
time_distributed_1 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 8857534 (33.79 MB)
Trainable params: 8857534 (33.79 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 10: Baseline LSTM Model Summary

3.8 Bidirectional LSTM

Our Bidirectional LSTM model is a little more intricate than our Bidirectional RNN in that this LSTM model should have the ability to remember more of the context of the sequence than the typical, even bidirectional, GRU. We also added an embedding layer as we hypothesized that our outcome would be better since our model was a little more complex. We then use an LSTM layer with the Bidirectional wrapper followed by two more Dense layers, one with a ReLu activation function and the other with a softmax activation function.



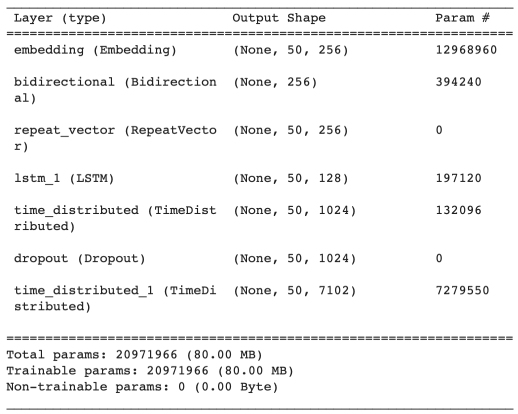
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 50, 300)	15198000
bidirectional (Bidirectional)	(None, 50, 256)	439296
time_distributed (TimeDistributed)	(None, 50, 1024)	263168
dropout (Dropout)	(None, 50, 1024)	0
time_distributed_1 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 23180014 (88.42 MB)
Trainable params: 23180014 (88.42 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 11: Bidirectional LSTM Model Summary

3.9 Encoder-Decoder LSTM

Similar to the other Encoder-Decoder based models, this model has two LSTM layers, one for the Encoder and one for the Decoder. We decided use the same structure as the RNN-based version of this model so we could make a direct comparison between the two different models. The only difference was an embedding layer for the LSTM since our input to the LSTM is a little different to the Encoder-Decoder RNN.



Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 50, 256)	12968960
bidirectional (Bidirectional)	(None, 256)	394240
repeat_vector (RepeatVector)	(None, 50, 256)	0
lstm_1 (LSTM)	(None, 50, 128)	197120
time_distributed (TimeDistributed)	(None, 50, 1024)	132096
dropout (Dropout)	(None, 50, 1024)	0
time_distributed_1 (TimeDistributed)	(None, 50, 7102)	7279550

Total params: 20971966 (80.00 MB)
Trainable params: 20971966 (80.00 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 12: Encoder - Decoder LSTM Model Summary

3.10 Bidirectional Encoder-Decoder LSTM With Attention

This model's architecture is similar to the Bidirectional Encoder-Decoder RNN with Attention for the same reason as the model before this one. We wanted to evaluate this model's performance and compare it to the similar models that preceded it. GRUs consist of an update gate and a reset gate which limit the model's ability to remember farther down the sequence. LSTMs have three gates: forget, input, and output, as well as a context vector that gets concatenated with the hidden states updates. LSTMs are generally known for being able to perform better on longer input sequences since they are able to leverage the context vector and hidden state updates which can hold significant knowledge of sequence.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder_1 (Encoder)         multiple                  497100

 decoder_1 (Decoder)         multiple                  1776500

=================================================================
Total params: 2273600 (8.67 MB)
Trainable params: 2273600 (8.67 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 13: Bidirectional Encoder - Decoder LSTM With Attention Model Summary

4 Results

As mentioned above, we evaluated our model with four different NLP metrics: BLEU-4, GLEU, METEOR, ChrF. We added accuracy in our metrics table to give some intuition about what the scores of some of the other evaluation metrics mean. It's important to note that the range of each metric is $\mathbb{R} \in [0, 100]$. Due to the quantity of models required and the time constraints we faced, we knew from the beginning that our models were not going be SOTA, but we came up with, designed, trained, and tested ten deep learning model architectures and received the results outlined in Figure 13. To our surprise, our best performing model was the Encoder-Decoder RNN with Attention with a BLEU score of 7.54 and a testing accuracy of 15.63 percent. We knew that this would be among the top performers since we added an entire attention layer in it; however, were surprised to see that the Encoder-Decoder LSTM with Attention did not outperform it. Our worst performing model

	BLEU	GLEU	METEOR	ChrF	Accuracy
baseline_rnn	0.733498	3.170400	6.414882	8.102562	4.027410
baseline_lstm	0.252936	1.207426	2.404387	3.373156	1.716696
bd_enc_dec_rnn	1.017243	3.764614	7.488236	9.537558	5.430774
bidirectional_rnn	1.009849	3.810797	7.372613	8.820557	5.125632
enc_dec_rnn	0.680936	2.987968	5.921060	7.523852	3.753773
rnn_embeddings	0.411181	2.256083	4.527484	5.752752	3.226130
enc_dec_rnn_attention	7.543138	12.604739	18.795220	21.985662	15.632754
enc_dec_lstm	0.914783	3.241193	6.543117	8.573683	3.842368
enc_dec_lstm_attention	5.147741	8.761840	14.083513	18.117795	11.414392
bidirectional_lstm	1.408349	4.519680	8.742193	10.700559	6.867096

Figure 14: Evaluation Metrics

was our Baseline LSTM with a BLEU score of 0.73 and a testing accuracy of 1.72 percent. We were surprised to see a better result with the basic GRU architecture versus the LSTM architecture. As mentioned in 3.10, intuition would have led us to believe that the LSTM finish at worst marginally better. The majority of predicted translations that came from these low scoring models were the words: 'the', 'of', 'and', 'witness', and <PAD>, which is the padding in a sentence when it's not long enough to fit the required max sequence length. Figure 14 is an example of a couple of our

```
Original Text:
if the moon becomes visible in on the day the will devour the wealth of the westland from
Translated Text:
if the moon becomes visible on the day the moon will be seen together reliable speech the land will become happy if the moon is surrounded by

Original Text:
if come close to the front of the moon and stand there the reign of the king will become long an enemy will attack but his downfall will take
Translated Text:
if the moon is surrounded by a halo and the moon and sun are seen together reliable speech the land will become happy the god will remember ad
```

Figure 15: Sample translation from our Encoder-Decoder RNN with Attention model

more accurate translations from our best model. These sentences are some of the few that were translated multiple times, we saw these same sentences throughout our translated lists from our models with attention. For the most part, our models predicted full sentences of the word 'the' and those mentioned above.

5 Test/Graphs/Discussions

We noticed from our results that the complexity of the model definitely matters, but from training we also learned that the complexity does not necessarily equate to optimal or better. The choice of our hyperparameters, such as unit size or weight initialization matters significantly. Most of our

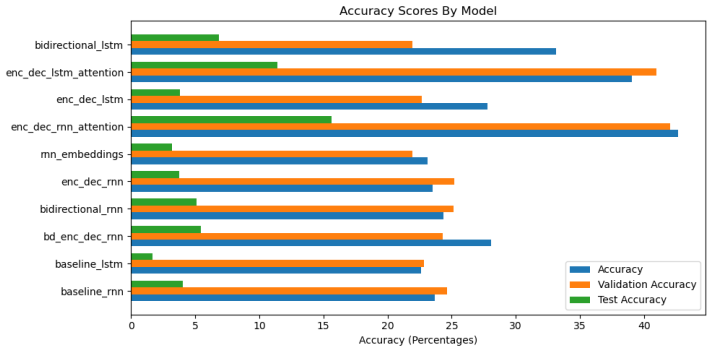
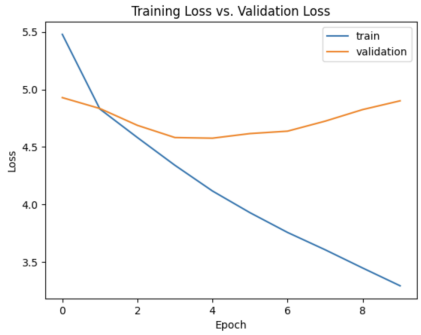


Figure 16: Comparison of Training, Validation, and Testing Accuracies

models' accuracies were generalizable across the training and validation sets with the exception of the bidirectional LSTM and the Encoder-Decoder LSTM, but unfortunately, none of our models generalized to the testing sets as seen in Figure 15. As mentioned in the Results, our models were outputting lists of single words or maybe two to three words different words. Based on our results from Figure 15, we can theorize that when our models were training and validating they were

(a) Bidirectional RNN



(b) Bidirectional LSTM

Figure 17: Loss graph comparison of Bidirectional models

over-fitting. Figure 16(a) shows the training and validation loss of the Bidirectional RNN model decending together nicely; however, the loss doesn't go anywhere near zero and the model suffers severely. The training loss of the Bidirectional LSTM was a lot better than the prior model's but the validation loss was worse. This didn't seem to matter to the testing set because the model's testing accuracy was the third highest after both of the models with attention layers added in. What is interesting is that we added an embedding layer to this LSTM model because we thought the result would be better despite having bad results from our RNN with the embedding layer. This also might have been a result of how we encoded and decoded our data, maybe a better encoding and decoding would have improved our results.

6 Constraints

One of our main constraints was access to affordable and powerful computing power. Our more intricate models were fairly complex but since we were training such long sequences even our simpler models were processor heavy, so whenever possible we would utilize the A100, V100, and T4 GPUs on Google Colab. We also have one member in our group with a Lenovo Windows OS that comes with a built-in Nvidia RTX A2000 GPU that was capable of running our more complex models locally. While it was able to run locally, it was very slow. The A100 GPU could train a model in a fraction of the time so it was a little time inefficient to use the A2000 for most of the training; however, it did get the job and it was free.

7 Standards

We utilized the Keras and TensorFlow libraries for tokenizing and training our models. Plain Keras was used for our simpler models so we were able to leave our data in terms of NumPy arrays; however, we utilized TensorFlow for our more complex models with attention and decided to use tensor objects to tokenize and train our models. For evaluation, we used the Natural Language Tooklkit library which allowed us to evaluate three of our metrics quite easily, but for the GTM metric we used the calculations found in the evaluation paper by Seugnjun[6]. We also used the

Pandas, NumPy, and MatplotLib libraries to organize our data and assist in understanding our results.

8 Comparison

Our best model is the Bidirectional Encoder-Decoder RNN with Attention. This model is different than most of our other models in both tokenization and structure. The structure is listed above in the corresponding section, but the key feature that made this model stand out from the others was the cross-attention layer added in during the decoding process. The top two models both had attention layers which made it quite evident that attention is very useful when creating models for machine translation. For a more in depth comparison of our models, refer to the Results section.

9 Limitations of the Study

Our biggest limitation of this study was time, if we had more time to work on this project with a lighter workload, i.e. not two to three other projects plus work, then we could have implemented more efficient models including both baseline and complex models. Another limitation we faced was our formatted data; we were training on long sequences of data where the sequence lengths were uneven among both input and output distributions. The longer lengths would be ideal for LLMs if we were able to better optimize over our tokenizers; however, due to time constraints this was not possible.

10 AI Fairness

We have three topics that we would like to address in terms of AI Fairness. Firstly, we want to address the bias in our training data. Machine translation systems are trained on text data, which can reflect societal biases present in the data. Biases related to gender, race, ethnicity, religion, nationality, and socio-economic status may inadvertently be perpetuated by the AI system if not properly addressed. Next we feel it's important to notice that our data consists of a language that is hugely underrepresented. Many languages and dialects are underrepresented in training datasets, leading to poorer translation quality for those languages. Lastly we want to address the topic of cultural sensitivity. Translations may inadvertently mistranslate culturally specific terms, idioms, or expressions, leading to misunderstandings or offense. A lack of cultural context awareness in translation systems can result in inappropriate or inaccurate translations, particularly in a multilingual setting like ours.

11 Future Work

We plan to continue this project after the completion of this class. Our goal is to most optimally design a model, most likely a Transformer, to accomplish our goal of making a highly efficient Neural Machine Translator. We also aim to focus on hyperparameter tuning and grid search to improve model performance on our baselines, but this will require more computational resources and more data. Ideally we will also have access to line and sentence sequence data. Unfortunately,

we did not have the bandwidth to accomplish this task during the duration of this course, but we have already begun the research for this future task.

12 Code Repository

https://github.com/Dbosty/CuneiTranslate/tree/master

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. "Attention Is All You Need". In: *arXiv:1706.03762v7* (12 Jun 2017).

[2] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473v7* (1 Sep 2014).

[3] Gai Gutherz, Shai Gordan, Luis Sáenz, Omer Levy, Jonathan Berant. "Translating Akkadian to English with neural machine translation". In: *PNAS Nexus, Volume 2, Issue 5* (May 2023).

[4] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[5] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *arXiv:1409.3215* (10 Sep 2014).

[6] Seungjun Lee et al. "A Survey on Evaluation Metrics for Machine Translation". In: 11.4 (2023).