DL Project Final Report: Shakespeare - English Sequence to Sequence Modeling

Morris Kraicer
Johns Hopkins University
mkraice1@jhu.edu

Riley Scott
Johns Hopkins University
rscott39@jhu.edu

William Watson
Johns Hopkins University
billwatson@jhu.edu

Abstract

We present our attempt to create an English-Shakespeare Sequence to Sequence Model, and its application to create a pseudo supervised style transfer system for text. We experimented with different Encoder-Decoder Architectures with and without Attention Mechanisms. We discuss our data procurement and processing to help facilitate learning, and present our experimental results and analysis.

1 Introduction

We applied a suite of Encoder-Decoder models with varying attention mechanisms (and lack of attention mechanisms) along with teacher forcing rates to a corpus of Modern English - Original Shakespeare data. Discussion of our data is in Section 2, our processing methods are described in Section 3. We used our neural machine translation systems to apply a supervised style transfer between the two forms of writing. More specifically, we tested models currently used in decoding languages (Section 4) and how they can be used in this capacity. We provide results for several model combinations and discuss the successes and failures of each system (Section 6). We also describe important implementation details in Section 5. Diagrams for our models can be found in the appendix.

2 Data

Most of our data came from previous alignment work by Xu et al.[9], in which two aligned corpora were procured from Sparknotes and Enotes. Both datasets, deriving from different websites, differed drastically. Sparknotes was more liberal in its translations, opting for reordering that differed drastically from the original plays, and was harder to learn. Enotes was a smaller corpus of 8 plays, but had less reordering, and was easier to train on. The data was aligned, and not all of the original lines from the plays are included (i.e. the sentences could not be aligned properly). [1]

Play	**Line Count**
Hamlet	2,010
Julius Caesar	1,201
Macbeth	1,085
Merchant of Venice	831
Midsummer Nights Dream	833
Othello	1,893
Romeo and Juliet	1,743
Tempest	769
Total	**10,365**

Figure 1: Line Counts for Shakespeare–English Corpus per Play (Enotes)

The vocabulary sizes are 9,004 source (original) words and 7,497 target (modern) words for the Sparknotes set. We had a total of 10,365 lines of Shakespeare plays to train on. There is a total word count of 233,282 words for the entire corpus (unprocessed). Sample data pairs (processed) are provided in Figure 2.

BLEU scores (disscussed in Section 5) for Original to Modern is 0.4455, and Modern to Original is 0.4465, indicating strong correlation between the two corpora.

[1] `https://github.com/cocoxu/Shakespeare`

#	Original Sentence	Modern Sentence
1	take me with you , take me with you , wife .	catch me , catch me , wife .
2	that 's fouler .	that 's even more evil .
3	perchance till after propn ' wedding - day .	maybe until after propn ' wedding - day .
4	propn truly knows that thou art false as hell .	propn truly knows that you are as false as hell .
5	then weep no more .	so stop crying .
6	of propn , a n't please your mastership .	of propn , if it pleases you , sir .
7	how ill this taper burns !	how badly this candle burns !
8	the charm 's wound up .	the charm s going to bring things to a head .
9	before the time be out ?	before the debt is paid ?
10	but soft !	quiet !

Figure 2: Sample Original-Modern Sentence Pairs, Processed

3 Preprocessing

We applied several techniques to improve the quality of our data and reduce the vocabulary size to train on.

Proper Nouns: Proper nouns are unique in both corpus, and have direct translations. In order to reduce vocabulary size and aggregate the learning of all proper nouns, we replace all proper nouns with the following token: propn. Thus our model should learn to map propn to propn, and can utilize the encoding to learn the most likely token following its usage. We use SpaCy's[2] nlp models to identify proper nouns in each sentence, and replace the tokens.

NLTK Tokenization: We use NLTK's `tokenize` module to further process our text data. We can use this module to split our strings into its component words, punctuation, contrations, etc. We can use this to easily split words and train on each token. We lower case all input before tokenizing.

SOS Tokens: During runtime, we encapsulate every sentence with two special tokens: SOS and EOS. The Start of Sentence token (SOS) signals the start of a sequence, and allows us to map our first real word to one that most likely starts a sentence, given the current hidden state from the encoder (and encoder outputs if attention is used).

EOS Tokens: We use the End of Sentence (EOS) token to signal the end of the sequence, i.e. when to stop the decoding process. In the batched version, the EOS token signals which tokens should be counted in the loss, while everthing after EOS is masked (see Section 5).

Data Split: We randomly shuffle and split the combined preprocessed dataset into three sets: Train, Dev, and Test. We opted for a 87.5/10/2.5 split to reduce the appearance of unkown tokens (UNK). This gives us a sizable training corpus to acutally learn, and a reasonable valdiation set to measure BLEU scores and loss. The small test set allows us to digest our results. Since the amount of Shakespeare data is limited, and will not increase in size, we can have small test sets and seek to overtrain the model.

File	Line Count
train.snt.aligned	9,069
dev.snt.aligned	1,036
test.snt.aligned	260
Total	**10,365**

Figure 3: Data Split for Shakespeare–English Corpus (Enotes)

4 Model Architecture

We developed several encoder-decoder style models with attention as used in Cho et al. [3] and Sutskever et al. [8].

4.1 Overview

Our approach is to use a state of the art neural MT system to provide a reasonable translation of the source sequence to a target language.

[2] `https://spacy.io`

NMT systems are comprised of two sub models: an Encoder and Decoder. The encoder is know as the *inference network* and encodes the input sequence to provide context to the decoder, or *generative network*. The decoder then uses the encoder's output to generate tokens corresponding to words in the target language. Attention mechanisms have shown to improve the generative network, as detailed in Bahdanau et al. [1], and further work from Luong et al. [5].

4.2 Encoders

The encoder, or *inference network*, receives an input token sequence $\vec{x} = [x_1, \ldots, x_n]$ of length n and processes this to create an output encoding. The result is a sequence $\vec{h} = [h_1, \cdots, h_n]$ that maps to the input sequence $\vec{x}$. The final hidden state is h_n.

Encoders share the same architecture except for the recurrent layer. For each input word x_j, the encoder looks up the associated word embedding, and runs the sequence through the recurrent layer. We define W_e as the word embedding, with each row corresponding to an input token.

$$\vec{h} = \begin{cases} \text{RNN}(W_e[\vec{x}]) \\ \text{GRU}(W_e[\vec{x}]) \\ \text{BiGRU}(W_e[\vec{x}]) \end{cases} \tag{1}$$

The output of the recurrent layer is our encodings, and we propagate the layer's last hidden state to the decoder. Theoretically, the hidden state should encode all the important information from the input sequence and pass it along to the decoder, but the encodings can be used with attention to provide better models. If the encoder type is bidirectional, then we return only the forward hidden state.

4.3 Decoders

The decoder, or *generative network*, receives the encoder outputs, the model's hidden state, and the last input token. On the first pass of the decoder, the hidden state is the encoder's final hidden state, and the input token is SOS.

We can describe our decoders in two ways: with and without attention. For our recurrent layers, we use the RNN and GRU layers. We cannot use a Bidirectional GRU because we do not know the full decoded sequence (and hence why we are decoding).

$$\tau_i = \begin{cases} W_e[t_i] & \textit{No Attention} \\ W_e[t_i] \parallel c_i & \textit{With Attention} \end{cases} \tag{2}$$

In the case of attention, we apply one of the attention schemes to the encoder output, given our current decoder's hidden state. We concatenate the attention results, known as a context vector c_i with our input embedding for an input token t_i. Without attention, we just use the input embedding and ignore the encoder outputs.

$$\mathbf{y} = W_d \cdot \begin{cases} \text{RNN}(\tau_i,\ s_{i-1}) \\ \text{GRU}(\tau_i,\ s_{i-1}) \end{cases} \tag{3}$$

We apply a recurrent layer to the attended tensor, using the hidden state provided. After applying a linear layer with weights W_d and applying a log softmax, we output our result for evaluation, which is the log probability distribution across the target vocabulary.

$$\log \sigma(\mathbf{y}) = \log \frac{\exp(y_i)}{\sum_j \exp(y_j)} \tag{4}$$

4.4 Attention

Attention mechanisms have been shown to improve sequence to sequence translations from Bahdanau et al. [1], and further work from Luong et al. [5] examines *global* vs *local* approaches to attention-based encoder-decoders. Common attention mechanisms are:

$$score(s_{i-1}, h_j) = \begin{cases} v_a^T \cdot \tanh(W_a[s_{i-1} \| h_j]) & \textit{concat} \\ s_{i-1}^T \cdot W_a \cdot h_j & \textit{general} \\ s_{i-1}^T \cdot h_j & \textit{dot} \end{cases} \tag{5}$$

where s_{i-1} is the previous decoder hidden state, and h_j is the jth encoder output. To compute scores, which are used to create attention weights, we apply a softmax:

$$a(s_{i-1}, h_j) = \frac{\exp(score(s_{i-1}, h_j))}{\sum_{j'} \exp(score(s_{i-1}, h_{j'}))} \tag{6}$$

Using these scores, we create a context vector c_i, which is just the batch matrix multiplication between our attention weights and the encoder outputs (i.e. a weighted sum of the encoder outputs).

$$c_i = \sum_{j'} a(s_{i-1}, h_{j'}) \cdot h_{j'} \tag{7}$$

We will focus on general attention and concat attention in our experiments.

Encoder	Decoder	Attention	Model Parameters
RNN	RNN	None	6,414,153
GRU	GRU	None	6,940,489
BiGRU	GRU	None	7,335,241
BiGRU	GRU	General	7,859,530
BiGRU	GRU	Concat	7,925,578

Figure 4: Total Model Parameters (for Enote dataset)

4.5 Teacher-Forcing

In terms of training, an encoder-decoder system can either accept the target token or the model's prediction as the next input during the decoding step. Teacher forcing is when we use the target token instead of the model's prediction and is shown to be favored during initial training iterations but should be tapered off to use the model's own predicitons. Continuous use of teacher forcing without backing off to the model's own tokens will exhibit instability in the translations as shown in Lamb et al. [4]

During training, we randomly determine whether to feed in the true target token or the model's decoded token to the decoder, and is configurable.

5 Implementation

Batching: In order to improve training speed and prevent noisy gradients, we introduced batching as recommended by Morishita et al. [6]. We batch by the sorting the source sequences by size in ascending order, and grouping sequences of equivalent sizes together. We pad each target sequence with the EOS token, up to the maximum target sequence length in the set. For our dataset, at a maximum batch size of 128, we created 151 training batches and 65 dev batches. We do not batch test samplesm.

GPU/CPU: Standard training can be done on a CPU, but neural networks have been shown to gain massive speedups. In order to improve training time, we have allowed an optional parameter to use a gpu. Our group has 3 GPUs (Morris: 1; Riley: 2). We distributed our tests across multiple GPUs for faster experimentation.

Masked NLL Loss: While we would normally use the *Negative Log Likelihood Loss* (NLL) to measure our error to backpropagate on, because of batching we must modify this loss to mask tokens that are past the target length for a decoded sequence. Hence we define a mask m such that for a target token index t_{ni} for the ith decoded token of the nth sequence in the batch:

$$m_{t_{ni}} = \begin{cases} 1 & \text{if } i < target\ length \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

We describe the masked loss for an input log probability vector x_n and true target token index t_{ni} as:

$$NLL(x_n, t_{ni}) = -x_{n,t_{ni}} \cdot m_{t_{ni}} \tag{9}$$

We average our loss across the batch.

BLEU Scores: We use the Bilingual Evaluation Understudy (BLEU) score as one of our criteria for the quality of model translations. Outputs are in the range [0, 1], and a BLEU score of 1 indicates a perfect translation to the reference texts. However, we do not need not obtain a perfect score in order to have a good translation model. A full description of how BLEU scores are calculated from Papineni et al. [7] can be found in Appendix A.4

6 Results: Shakespeare-English

Our results demonstrate how different models and teaching methods affect the final translations. In Figure 5 below, we can see the final BLEU scores for our models.

RNN: The bare bones RNN had almost no capacity to learn and suffers from issues such as the vansihing gradient problem and long term dependencies. Since many of our sentences were long, this was problematic. The most distingiushing feature of the RNN translations is the overuse of repeated punctuation.

Model	Source BLEU	Target BLEU
RNN	0.0000	0.0000
GRU	0.0237	0.0238
BiGRU	0.0297	0.0336
BiGRU + General	0.0291	0.0333
BiGRU + General + 50% TF	0.0264	0.0330
BiGRU + General + 100% TF	0.0348	0.0395
BiGRU + Concat	0.0953	0.0842
BiGRU + Concat + 50% TF	0.1505	0.1349
BiGRU + Concat + 100% TF	0.1153	0.1042
Best Model	**0.1647**	**0.1450**
Original-Modern	**0.4465**	**0.4455**

Figure 5: Model BLEU Results for Experiments. Source BLEU compares results to the source (original) sentences. Target BLEU compares results to the target (modern) sentences. We aim for high Target BLEUs.

GRU and Bidirectional GRU: The GRU showed better promise in acheiveing higher bleu scores, as seen in Figure 5, since GRUs migitgate the vanishing gradient problem and have a higher capacity. The Birdirectional GRU achieved slightly better results, most likely from incorporating both a forward and backward context to the sequence. In addition, the bidirectional has two sublayers, thus doubling the GRU capacity to learn.

Attention Mechanisms: Although our best results were from a model trained with concat attention, we can see that the model trained with general attention actually worked well on shorter sentences (but concat still outperformed general). An explanation for this is that Luong et al. [5] attention mechanisms work best in a local setting. This means that either the sentnece is short, or the model only looks at a small window of the full sequence. It is reasonable to assume that since we looked at the global context of our source encodings, the general attention failed whereas the Bahdanau [1] concat attention performed better overall.

Teacher Forcing: The 'best model' score came from our BiGRU architecture with concat attention and 100% teacher forcing. Looking at the scores, we can actually see that our model trained with 50% teacher forcing did better than the one with 100%, but our final best results were from the one with 100% and after many iterations with random initializations and keeping the trained model that worked best. This is important to note since it demonstrates how randomness in initializations can greatly affect the final outcome. If we had more time and resources, we would train a model that starts with 100% teacher forcing and slowly taper down as training progresses.

Best Model: Our best model, as selected by test BLEU score, was the BiGRU architecture with concat attention and 100% teacher forcing, after training for 500 iterations. This model acheive a target BLEU score of 0.1450. The next best model was the BiGRU with concat attention and 50% teacher forcing, with target BLEU score of 0.1349.

Final Analysis: This shows that attention is necessary for good results, and that teacher forcing helps the model achieve better translations by feeding in the true target tokens. However, these models can be trained longer, and the effects of random initialization is not clear. In addition, more training dynamics can be incorporated to provide deeper insight into this problem. See Appendix B for sample translations.

7 Key Takeaways

General vs Local Attention: Our attention models use the *global* approach which attends to all the source embeddings at once. However, Luong et al. [5] showed that by using a *local* approach that attends to only a subset of the encodings, a nerual encoder-decoder system could make effective gains in BLEU scores. In our experiments, both attention models used the *global* approach, and it is clear why general failed to do well comapred to the Bahdanau concat attention. Luong states that *global* attention fails on long sequences whereas a local approach is more effective.

The *local* approach takes source embeddings within a window, computes the attention, and weights the attention probabilities by a guassian distribution.

Teacher Forcing Impact: Teacher Forcing was required to make effective gains in our attention models. However, our results are inconclusive on how much teacher forcing should be applied. According to Lamb et al. [4], teacher forcing acts as a regularizer for these systems. In addition, Bengio et al. [2] describes an annealing process, changing the task from an easy one with target tokens, to a more realistic approach of feeding in the model's prediction.

Random Initialization: As our results show, different initializations can lead to different BLEU scores, and different generalization performance. We saw this between our concat models that used teacher forcing.

GPU vs CPU Training Times: Training on a GPU yielded significant performance increases, allowing us to train multiple models quickly. Our model with the largest capacity took 11-13 minutes on a standard CPU, but only 111 seconds on a GPU for 2 full epochs.

8 Conclusion & Future Work

If given more time, obvious extensions to our existing experiments would included:

Batch Annealing: We wanted to study the effects of adaptive batch sizes, where training would begin with a small batch size and progressively increase to very large sizes. This would allow for stochastic updates in the begining before approximating full gradients in later iterations. We think this would have improved training and provided better results.

English to Shakespeare Modeling: We wanted to also train a reverse model for English to Shakespeare that applies a translation between English sentences and turns them into Shakespeare sentences. This is a weak form of style transfer masked as a translation problem. However, more work needs to be done before we find the best architecture.

Local Attention with Window: As discussed in Section 7, incorporating a window to allow for *local* attention over *global* could results in better models. General might perform better in a *local* context over a *global* approach.

Data Processing: We may want to look into other processing pipelines, such as the ones found in SMT platforms Moses or GIZA++. In addition, we may want to look into BPE tokenization to see if it reduces our vocabulary size. BPE tokenization segments text into subword units, and has been found to significantly reduce vocabulary sizes in some languages. More time could be spent on refining our current dataset, and procuring more data to train on. Finally, we should sample test sentences such that no unkown words are in it, improving test results.

Implementation: Our codebase for this project can be found here.

9 Advice to Future Students

It's important for students to understand that most ML problems rely on vast amounts of data. Now, deep learning systems have been shown in practice to handle raw data effectively, but time should still be spent on refining the data. We had this issue between two corpora that drastically differed in reordering of text, making one more difficult to learn.

Additionally, strong foundations of linear algebra, multivariate calculus, optimization and probability are required to understand how these models are constructed and trained. Multivariate calculus and optimization is fundamental to backpropagation and gradient descent. Probability gives us maximum likelihood estimation, the basis for most loss functions, and influences our model's approaches to predicition. Linear Alegbra is essential to expressing complex equations into simple statements, and understanding the fundamentals of neural networks and how they transform inputs.

The more advanced models such as VAEs, RBMs, and generative models become accessable with the prerequisite knowledge. Therefore, students should be taught these subjects with respect to neural networks as early as possible. However, the instructors should attempt to teach neural networks in a non-standard way, as simple classification and regression results in rigid thinking oh how models can be designed, and discussion on how models can be trained in non-standard ways, such as NMT models or VAEs, would be great.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099, 2015.

[3] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[4] A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609, 2016.

[5] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[6] M. Morishita, Y. Oda, G. Neubig, K. Yoshino, K. Sudoh, and S. Nakamura. An empirical study of mini-batch creation strategies for neural machine translation. *arXiv preprint arXiv:1706.05765*, 2017.

[7] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[8] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[9] W. Xu, A. Ritter, B. Dolan, R. Grishman, and C. Cherry. Paraphrasing for style. In *COLING*, pages 2899–2914, 2012.

A Additional Information

A.1 RNN

The simplest way to process sequences of inputs is to use recurrent layers, which accept an input and previous hidden state to update its output. Our baseline models incorporate the *Elman RNN*, the simplest recurrent layer.

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \tag{10}$$

However, the layer is prone to the vanishing gradient problem on large sequences. In addition, *Elman RNNs* have the lowest capacity to learn.

A.2 GRU

Improvements have been made to the baseline RNN to increase capacity and improve gradient flow over long sequences. *Long Short-Term Memory* (LSTM) and *Gated Recurrent Units* (GRU) are two of the most popular improvements to recurrent networks. Both solve the vanishing gradient problem found in traditional RNN layers, and improve a model's capacity to learn.

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t \circ (W_{hn}h_{t-1} + b_{hn})) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \tag{11}$$

For our models, we use GRUs, as they have been shown to perform as well as LSTMs but with less parameters.

A.3 Bidirectional GRU

GRUs, and RNNs in general, only process sequences in a forward direction. However, encoding both the preceding and succeeding input would be beneficial to learning the context of a word in its local vicinity. Bidirectional layers allow models to encode inputs in both the forward and reverse direction. This is a simple, yet poweful, extension to uni-directional recurrent layers.

$$\begin{aligned} \overrightarrow{h_f} &= \text{GRU}(\overrightarrow{input}) \\ \overleftarrow{h_b} &= \text{GRU}(\overleftarrow{input}) \\ h_o &= \overrightarrow{h_f} \parallel \overrightarrow{h_b} \end{aligned} \tag{12}$$

Bidirectional layers run two separate recurrent layers, one on the forward input sequence; the other on the reversed input. Thus we get two output tensors, one for each direction. We reverse the backward tensor, and stack the two output tensors together. Thus our final tensor encodes the forward and backward context.

A.4 BLEU Scores

We use the Bilingual Evaluation Understudy (BLEU) score as one of our criteria for the quality of model translations. Outputs are in the range [0, 1], and a BLEU score of 1 indicates a perfect translation to the reference texts. We do not need not obtain a perfect score in order to have a good translation model. BLEU scores are calculated for individual sentences and averaged across the whole corpus. Intelligibility and grammatical correctness are not taken into account when calculating BLEU scores. According to the original paper by Papineni et al. [7], BLEU scores are computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{13}$$

where p_n is the geometric average of the modified n-gram precisions and BP is the brevity penalty, explained below. ([7] uses $N = 4$, uniform weights $w_n = \frac{1}{N}$.) The brevity penalty BP is used for translations that are too short compared to the reference. BP is defined as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{14}$$

with effective reference corpus length r and candidate translation length c. BLEU scores are non-differentiable, and therefore we cannot directly optimize on this metric, but can use it to enhance our understanding regarding the quaity of our models given the data.

B Model Results

Model	**Sentence**
Source	**double , double , toil and trouble ; fire burn and cauldron bubble .**
RNN	damned as you are , you have cast the spell , the [followed by 490 commas]
GRU	double , double , toil and ; ; and burn burn burn burn caldron bubble .
BiGRU	double , double , toil and ; ; burn , , , and caldron bubble . .
BiGRU + General	double , double , toil , and ; , , burn , , bubble bubble .
BiGRU + General + 50% TF	double , double , toil and trouble ; fire , burn ; and caldron , bubble .
BiGRU + General + 100% TF	double , double , toil and trouble ; fire , burn ; and caldron , bubble .
BiGRU + Concat	double , double , toil and trouble ; , fire burn ; and caldron bubble bubble .
BiGRU + Concat + 50% TF	double , double , toil and trouble ; fire , burn ; and caldron , bubble .
BiGRU + Concat + 100% TF	double , double , toil and trouble ; fire , burn ; and caldron , bubble .
Best Model	**double , double , toil and trouble ; fire , burn ; and caldron , bubble .**
Target	**double , double , toil and trouble ; fire , burn ; and caldron , bubble .**

Figure 6: Model Results for Selected Source-Target Pair: #1

Model	**Sentence**
Source	**tonight , my lord ?**
RNN	what you you ,
GRU	how , my lord ?
BiGRU	my , my lord ?
BiGRU + General	, my lord lord ?
BiGRU + General + 50% TF	my lord lord ?
BiGRU + General + 100% TF	brother , my lord ?
BiGRU + Concat	tonight , my lord ?
BiGRU + Concat + 50% TF	three , , my lord ?
BiGRU + Concat + 100% TF	tonight , my lord ?
Best Model	**tonight , my lord ?**
Target	**tonight , my lord ?**

Figure 7: Model Results for Selected Source-Target Pair: #2

Model	**Sentence**
Source	**what say you , propn ?**
RNN	what , you ,
GRU	what do you , propn propn ?
BiGRU	what do you say , propn ?
BiGRU + General	what do you say , propn ?
BiGRU + General + 50% TF	what do you say , propn ?
BiGRU + General + 100% TF	what do you say , propn ?
BiGRU + Concat	what did you say , propn ?
BiGRU + Concat + 50% TF	what do you say , propn ?
BiGRU + Concat + 100% TF	what do you say , propn propn ?
Best Model	**what do you say , propn ?**
Target	**what do you say , propn ?**

Figure 8: Model Results for Selected Source-Target Pair: #3

Model	**Sentence**
Source	**propn thou not see her paddle with the palm of his hand ?**
RNN	do n ' t
GRU	will n ' t storms ; his his his the ? ?
BiGRU	propn ' n ' t with you like the the the the hand hand ?
BiGRU + General	does t you t his his his his his all ?
BiGRU + General + 50% TF	propn , remember you in the hand of night of great hand ?
BiGRU + General + 100% TF	might do n t see the his dying hand , might see about the ?
BiGRU + Concat	do you ' t that you speak with the with such of of hand hand ?
BiGRU + Concat + 50% TF	have n ' t see her with the same of of his his hand ?
BiGRU + Concat + 100% TF	do n ' t you see her for the scripture of his body ?
Best Model	**did n ' t go see her close with the same of his hand ?**
Target	**did n t you see her play with the palm of his hand ?**

Figure 9: Model Results for Selected Source-Target Pair: #4

Model	**Sentence**
Source	**some wine , ho !**
RNN	what , ho !
GRU	what , , , !
BiGRU	and , ho !
BiGRU + General	give , oh ! !
BiGRU + General + 50% TF	oh , oh , hello !
BiGRU + General + 100% TF	some three , a blessing in like a !
BiGRU + Concat	some ! , ho !
BiGRU + Concat + 50% TF	let me fight !
BiGRU + Concat + 100% TF	some whiskey , hello !
Best Model	**some wine , ho !**
Target	**some wine , ho !**

Figure 10: Model Results for Selected Source-Target Pair: #5

Model	**Sentence**
Source	**down , strumpet !**
RNN	oh ,
GRU	hey , hey prostitute
BiGRU	prostitute , ! !
BiGRU + General	no , !
BiGRU + General + 50% TF	propn , !
BiGRU + General + 100% TF	prostitute !
BiGRU + Concat	down , , prostitute !
BiGRU + Concat + 50% TF	peace , prostitute !
BiGRU + Concat + 100% TF	down , prostitute !
Best Model	**down , prostitute !**
Target	**down , prostitute !**

Figure 11: Model Results for Selected Source-Target Pair: #6

Model	**Sentence**
Source	**let each man render me his bloody hand .**
RNN	and , , the , , , , ,
GRU	his me his him him a , him revenge . .
BiGRU	then , let his ; me ' m me me .
BiGRU + General	let me , where where to his . .
BiGRU + General + 50% TF	let me leave his hand away .
BiGRU + General + 100% TF	let me me me , good hand hand , but hand hand hand me .
BiGRU + Concat	let give they give me me his hand hand .
BiGRU + Concat + 50% TF	let the man from give me his honorable .
BiGRU + Concat + 100% TF	let go get saying any man .
Best Model	**let me shake his beard hand .**
Target	**let each man give me his bloody hand .**

Figure 12: Model Results for Selected Source-Target Pair: #7

C Model Architecture Diagrams

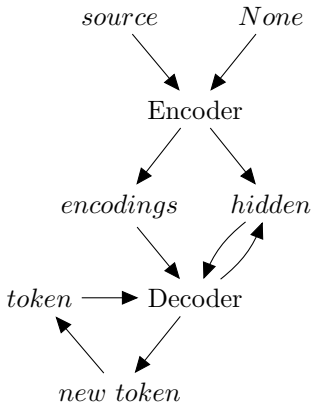

Figure 13: Model Architecture Overview for Encoder-Decoder.

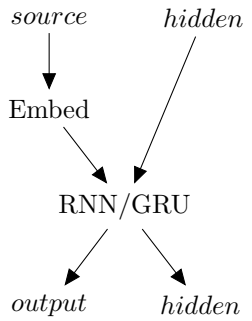

Figure 14: Model Architecture for Encoder.

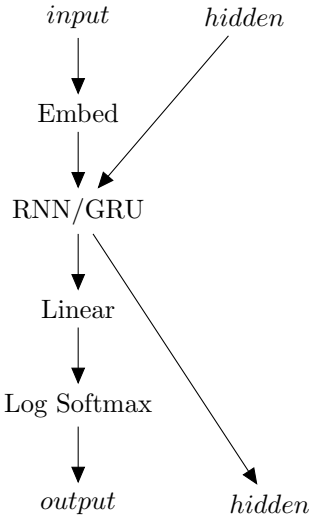

Figure 15: Decoder with No Attention.

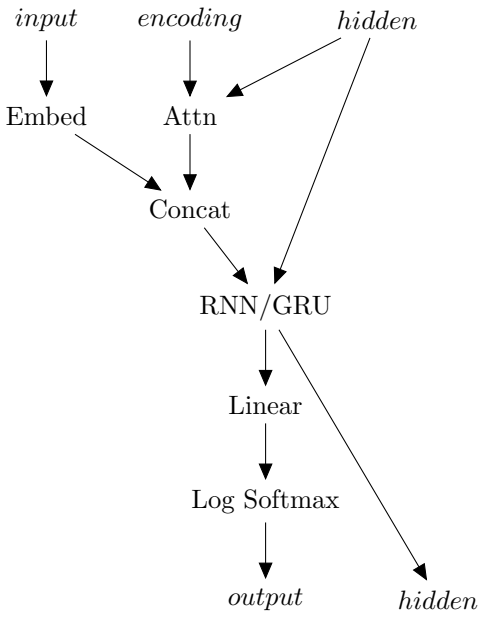

Figure 16: Decoder with Attention.

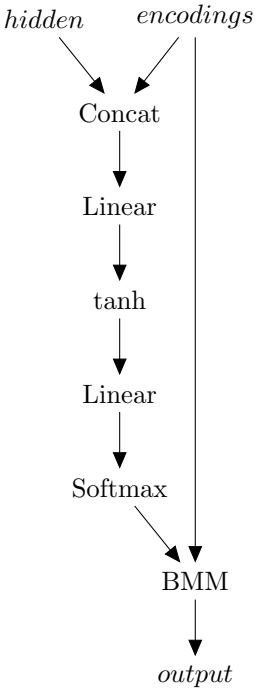

Figure 17: Concat (Bahdanau) Attention Layer.

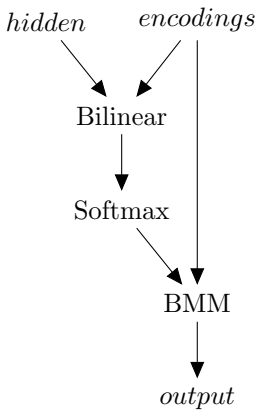

Figure 18: General Attention Layer.