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.

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. Our final results are described in Section 7 for our style transfer system. 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.[7], 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.

3 Preprocessing

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

[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

3.1 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.

3.2 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.

For instance, the modern sentence from Hamlet *there's rue for you, and here's some for me.* will be tokenized into: *there 's rue for you , and here 's some for me .*

Additionally, contrations are split: *isn't he honest?* tokenizes to *is n ' t he honest ?*

We can use this to easily split words and train on each token. We lower case all input before tokenizing.

3.3 SOS/EOS 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).

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

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 5.3.1).

3.4 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 validiation 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 models to create our pseudo style transfer system, incorporating context, attention, and training methods. It encorporates an encoder-decoder

style model as used in Cho et al. [2] and Sutskever et al. [6].

4.1 Overview

4.1.1 RNN

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

4.1.2 GRU

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t \circ (W_{hn}h_{t-1} + b_{hn})) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \tag{2}$$

4.1.3 Bidirectional GRU

$$\begin{aligned} \overrightarrow{h_f} &= \text{GRU}(\overrightarrow{input}) \\ \overleftarrow{h_b} &= \text{GRU}(\overleftarrow{input}) \\ h_o &= \overrightarrow{h_f} \parallel \overrightarrow{h_b} \end{aligned} \tag{3}$$

4.2 Encoders

4.3 Decoders

4.4 Attention

4.5 Teacher-Forcing

5 Implementation

5.1 Batching

5.2 GPU/CPU

5.3 Metrics

5.3.1 Masked NLL Loss

5.3.2 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. [5], BLEU scores are computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{4}$$

where p_n is the geometric average of the modified n-gram precisions and BP is the brevity penalty, explained below. ([5] 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:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{5}$$

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.

6 Results: Shakespeare-English

7 Style-Transfer: English-Shakespeare

8 Conclusion & Future Work

9 Architectures

9.1 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 $\vec{h} = [h_1, \cdots, h_{T_x}]$ that maps to the input sequence $\vec{x}$.

9.1.1 Baseline RNN Encoder

For the baseline encoder, we implemented a simple Embedding + RNN encoder. This accepts an input sequence $\vec{x}$ and encoders the sequence using the lookup embeddings and forward context.

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{t-1} + b_{hh}) \tag{6}$$

Howver, this is the simplest model, prone to the vanishing gradient problem on large sequences. In addition, this model has the lowest capacity to learn. Nonetheless, we will present results for our baseline.

9.1.2 GRU Encoder

An obvious improvement to our encoding scheme would be to replace the RNN layer with a GRU. A GRU encodes a forward sequence using more complex equations to improve capacity and learning.

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t \circ (W_{hn}h_{t-1} + b_{hn})) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \tag{7}$$

GRUs help with vanishing gradients, increases our models capacity to learn, and uses less parameters than the LSTM layer. Hence, for our purposes, we used a GRU over the LSTM.

9.1.3 Bidirectional GRU Encoder

An extension to our encoding scheme will consider the full context of words immediately before an after it. This is done by running a GRU layer on the forward and backward sequence and combining each tensor. Hence we use a bidirectional GRU as laid forth in Bahdanau et al. [1]

$$\begin{aligned} \overrightarrow{h_f} &= \text{GRU}(\overrightarrow{input}) \\ \overleftarrow{h_b} &= \text{GRU}(\overleftarrow{input}) \\ h_o &= \overrightarrow{h_f} \parallel \overrightarrow{h_b} \end{aligned} \tag{8}$$

PyTorch concatenates the resulting tensors, doubling the hidden output size of a normal GRU. Other libraries allow for concatenatation, averaging, and summing. We use PyTorch's implementation of a bidirectional GRU.

9.1.4 Implementation

All encoders share the same model architecture, except for the recurrent layer. We use PyTorch's implementation for the RNN, GRU, and Bidirectional GRU. We also add embedding layers. PyTorch's recurrent layers naturally support multiple layers and dropout, and can be set through CLI args `--num-layers` and `--lstm-dropout`. The hidden size is set by `--hidden-size`. The defaults are 1, 0.1, and 256, respectively.

9.2 Decoders

We will describe the decoder algorithm, and experiment with two different recurrent layers: RNN and GRU. We cannot use a Bidirectional GRU because we do not know the full decoded sequence (and hence why we are decoding).

Decoders take in the last translated token, starting with an SOS token on a new batch. It applies an embedding layer, followed by an optional dropout.

We then have two options:

1. Attention (General, Concat, etc.)
2. No Attention

In the case of attention, we apply one of the attention schemes (described in the next section) to the encoder output, given our current decoding hidden state. We concatenate the attention results with our input embedding.

Without attention, we just use the input embedding and ignore the encoder outputs.

We apply a recurrent layer to the attended tensor, using the hidden state provided. After applying a linear layer and log softmax, we output our result for evaluation.

9.3 Attention Mechanisms

Attention mechanisms have been shown to improve sequence to sequence translations from Bahdanau et al. [1], and further work from Luong et al. [4] examines global vs local approached to attention-based encoder-decoders. Common attention mechanisms are:

$$score(h_t, h_s) = \begin{cases} v_a^T \cdot \tanh(W_a[h_t \| h_s]) & concat \\ h_t^T \cdot W_a \cdot h_s & general \\ h_t^T \cdot h_s & dot \end{cases} \tag{9}$$

where h_t is the current target hidden state, and h_s is the encoder output. To compute scores, which are used to create attention weights, we apply a softmax:

$$a_t(s) = \frac{\exp(score(h_t, h_s))}{\sum_{s'} \exp(score(h_t, h_{s'}))} \tag{10}$$

Using these scores, we create a context vector, which is just the batch matrix multiplication between our attention weights and the encoder outputs. We will focus on general attention and concat attention in our experiments.

#	Encoder	Decoder	Attention	Teacher Forcing (Percent)
1	RNN	RNN	None	None
2	GRU	GRU	None	None
3	Bidirectional GRU	GRU	None	None
4	Bidirectional GRU	GRU	Concat	None
5	Bidirectional GRU	GRU	General	None
6	Bidirectional GRU	GRU	Concat	0.5
7	Bidirectional GRU	GRU	General	0.5
8	Bidirectional GRU	GRU	Concat	1.0
9	Bidirectional GRU	GRU	General	1.0

Figure 4: Planned Model Experiments (Subject to change depending on results)

9.4 Teacher Forcing

In terms of training, an encoder-decoder system can either accept the target token or the model's prediction as input during the decoding step. When we use the target token, this is known as teacher forcing, and is shown to be favored during initial training iterations, but should be backed off to use the model's own predicitons, as it will exhibit instability in the translations otherwise, as written in Lamb et al. [3]

We hope to build in a system to decay the teacher forcing percentage over time, instead of our current implementation that checks a random number againist the hyperparameter. However, we can bypass this effectively by reloading a model and changing the teacher forcing parameter provided.

10 Planned Experiments

We plan to experiment with several model permutations, as outlined in Figure 4. We do not test every permutation since that would take too much time, and have selected several runs that would provide us enough results to determine the best model for this problem.

11 Roadblocks and Problems

We have hit two major roadblocks since our proposal, and discuss our apporaches to mediating the issues.

11.1 Data Learnability

In our first tests, we used data procured from previous alignment work by Xu et al. [7] There were two parallel datasets, one from Sparknotes, the other from enotes. We originally planned to use the Sparknotes version, as the data set was larger (21079 sentence pairs). However, when training, the models had difficulty accounting for the reordering Sparknotes used in their translations. Hence, we decieded to switch to enotes, which is a smaller (10365 sentence pairs), but more aligned corpus. We theorize that data that has less reordering will converge faster to a desired result. Since the dataset is smaller, we use less batches, and training time trivially improves. We hope for decent results by switching.

11.2 GPU/CPU Training Times

Systems like these take a long time to learn the data, and one of our major roadblocks is training time. For a normal CPU run, 2 epochs takes 11-30 minutes, depending on computer specs. This would put our experiments total running time at over 2-3 days per experiment (18-27 days total). This does not give us much leeway to verify our apporach. Hence, we describe in Section **??** our intial resutls in GPU compatability and speedups. These timings were done on the Sparknotes dataset.

12 Additional Implementation Details

12.1 Batching

In order to faciliate faster training, and less noisy gradients, we felt it imperative to introduce batching of sentences. We batched similiar sentences according to source sentence length (encoder input). This allows us to reduce the number of batches to loop through and take advantage of torch operations.

#	Task	Status	Team Member
1	Data Procurement	DONE	All
2	Preprocessing	DONE	Riley, Bill
3	RNN Encoder	DONE	Riley
4	GRU Encoder	DONE	Morris
5	Bidirectional GRU	DONE	Morris
6	RNN Decoder	TESTING	Morris
7	GRU Decoder	TESTING	Bill
8	Attention Models	TESTING	Bill
9	Teacher Forcing	DONE	Bill
10	Vocabulary Building	DONE	Bill
11	Train/Eval Support Code	DONE	Bill
12	GPU Support	DONE	Riley
13	Batching	DONE	Bill
14	Experiments	IN PROGRESS	Morris, Riley

Figure 5: Current Progress

12.2 GPU Compatability

Initial training is slow on a cpu, with a Bidirectional GRU Encoder + GRU Decoder + Concat Attention estimated at 11-30 minutes per 2 epochs, varying on batch size (32 and 128 tested). In order to improve training time, we have allowed an optional parameter to use a gpu. Inital run on batch size 128 yielded a training speed of 111 seconds per 2 epochs. Hence our experiments will take only 4-6 days, and can be distributed across multiple GPUs for even less time.

Our group has an estimated 3 GPUs (Morris:1; Riley:2). This will help us distribute our experiments across several computers and allow us time to adjust experiments based upon our initial findings. If need be, we will us Google credits to fund more experiments.

13 Current Status and Expectation

See Figure 5 for our current standing on this project. Our notation is as follows:

1. DONE - Fully implemented, tested, no further work needed
2. IN PROGRESS - Currently being implemented or worked on. Some inital results but more work is required for full functionality.
3. TESTING - Fully functional, but currently being tested for bugs, etc.

While most of the coding is done, and the last few model architectures are being tested, we estimate the longest task to be completed is model experimentation, and may take the next two-three weeks to complete.

We will automate all planned experiments, examine the results, and use the best model as our style-transfer. We will then begin our final training session on both directions and provide results. We hopefully expect training of our experiments completed by the end of Thanksgiving break, and can pick a model to perform style-transfer for English-Shakespeare examples.

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] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

A Model Architecture Diagrams

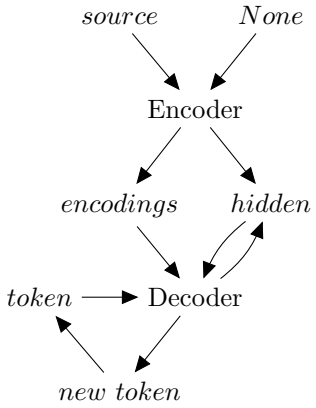

Figure 6: Model Architecture Overview for Encoder-Decoder.

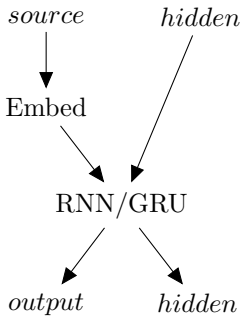

Figure 7: Model Architecture for Encoder.

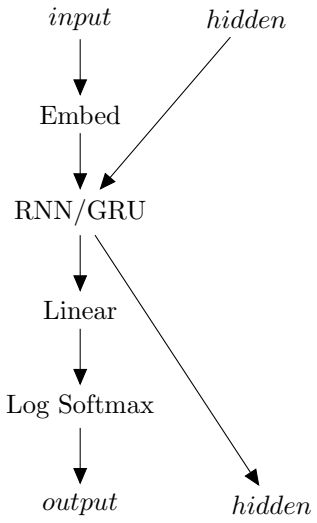

Figure 8: Decoder with No Attention.

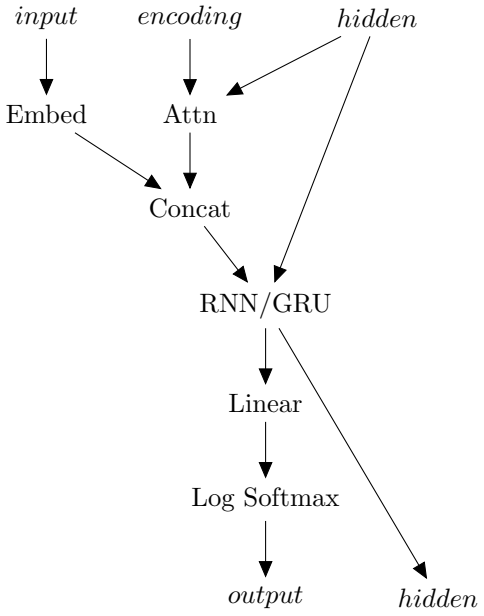

Figure 9: Decoder with Attention.

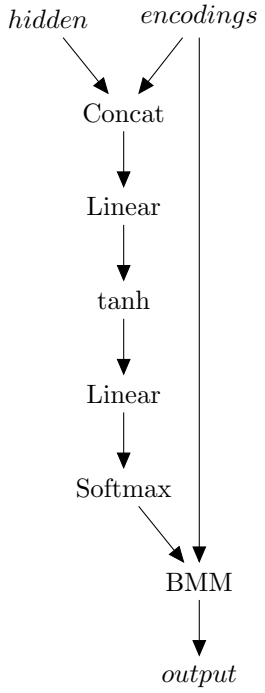

Figure 10: Concat (Bahdanau) Attention Layer.

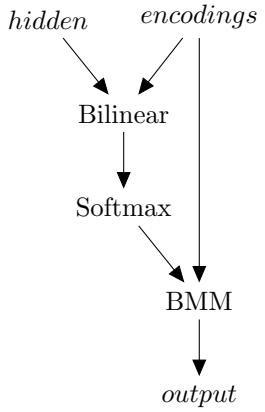

Figure 11: General Attention Layer.