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.[8], 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. [7].

4.1 Overview

Our idea is inspired by reducing our style-transfer system to a translation problem. Hence our approach to provide state of the art neural MT systems to provide a reasonable transfer of the target styling for a given source sequence.

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. [4].

We describe the underlying components used to build these sub-networks, and their advantages and disadvantages.

4.1.1 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.

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

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

4.1.2 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.

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

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

4.1.3 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.

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

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.

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.

$$\vec{h} = \begin{cases} \text{RNN}(W_e[\vec{x}]) \\ \text{GRU}(W_e[\vec{x}]) \\ \text{BiGRU}(W_e[\vec{x}]) \end{cases} \tag{4}$$

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.

We use three types of encoders: RNN, GRU, and the Bidirectional GRU (BiGRU).

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).

$$\tau_i = \begin{cases} W_e[t_i] & \textit{No Attention} \\ W_e[t_i] \parallel c_i & \textit{With Attention} \end{cases} \tag{5}$$

In the case of attention, we apply one of the attention schemes (described in the next section) 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.

$$\mathbf{y} = W_d \cdot \begin{cases} \text{RNN}(\tau_i,\, s_{i-1}) \\ \text{GRU}(\tau_i,\, s_{i-1}) \end{cases} \tag{6}$$

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.

$$\log \sigma(\mathbf{y}) = \log \frac{\exp(y_i)}{\sum_j \exp(y_j)} \tag{7}$$

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. [4] examines global vs local approaches to attention-based encoder-decoders. Common attention mechanisms are:

$$score(s_{i-1}, h_j) = \begin{cases} v_a^T \cdot \tanh(W_a[s_{i-1} \| h_j]) & \textit{concat} \\ s_{i-1}^T \cdot W_a \cdot h_j & \textit{general} \\ s_{i-1}^T \cdot h_j & \textit{dot} \end{cases} \tag{8}$$

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:

$$a(s_{i-1}, h_j) = \frac{\exp(score(s_{i-1}, h_j))}{\sum_{j'} \exp(score(s_{i-1}, h_{j'}))} \tag{9}$$

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.

$$c_i = \sum_{j'} a(s_{i-1}, h_{j'}) \cdot h_{j'} \tag{10}$$

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

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. 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. Continuous use of teacher forcing without backing off to the model's own decoded tokens will exhibit instability in the translations as shown in Lamb et al. [3]

During training, we randomly determine whether to feed in the true target token instead of the generated token to the decoder. This is a configurable parameter that deterimes the threshold to allow teacher forcing, where 0 means no teacher forcing and 1 is constant teacher forcing.

5 Implementation

5.1 Batching

In order to improve training speed and prevent noisy gradients, we introduced batching for our model. As recommended by Morishita et al. [5], we batch by the sorting the source sequences by size in ascending order, and grouping sequences of equivalent sizes together. We have the option to limit the maximum size of the batches, which is currently defaulted to 128. For batching the target sequences, we pad each sequence with the EOS token, up to the maximum target sequence length in the set.

This allows us to run multiple sequences through the model simultaneously, and build up decoded sequences

for many sentences at a time. This also improves learning by averaging the gradients as they are backpropagated.

For our dataset, at a maximum batch size of 128, we created 151 training batches and 65 dev batches.

5.2 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, if system capable.

Our group has 3 GPUs (Morris: 1; Riley: 2). We distributed our tests across multiple GPUs for faster experimentation.

5.3 Metrics

Our model uses two main metrics to describe the quality of our system.

5.3.1 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.

$$NLL(x_n, y_n) = -x_{n,y_n} \tag{11}$$

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:

$$m_{t_{ni}} = \begin{cases} 1 & \text{if } i < \textit{target length} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

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

$$NLL(x_n, t_{ni}) = -x_{n,t_{ni}} \cdot m_{t_{ni}} \tag{13}$$

We average our loss across the batch

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. [6], BLEU scores are computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{14}$$

where p_n is the geometric average of the modified n-gram precisions and BP is the brevity penalty, explained below. ([6] 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:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{15}$$

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

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] 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.

[6] 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.

[7] 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.

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

A Model Architecture Diagrams

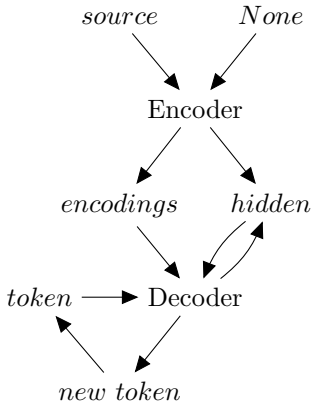

Figure 4: Model Architecture Overview for Encoder-Decoder.

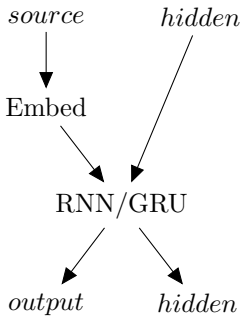

Figure 5: Model Architecture for Encoder.

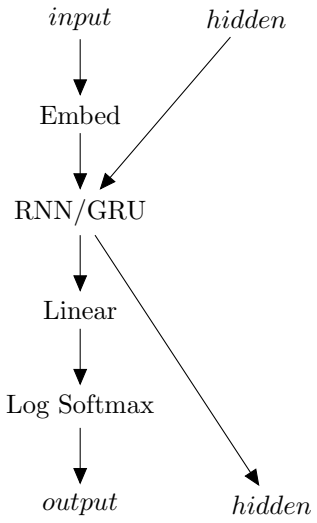

Figure 6: Decoder with No Attention.

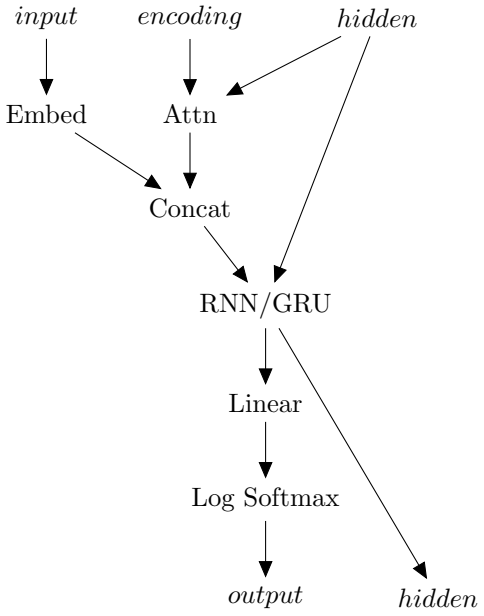

Figure 7: Decoder with Attention.

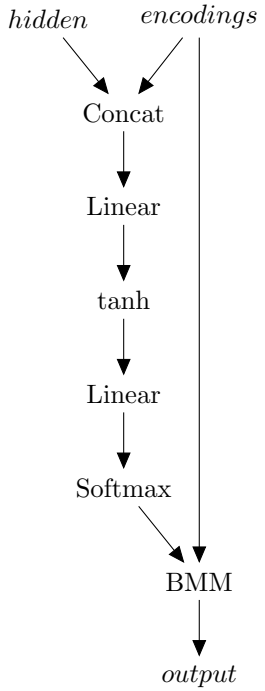

Figure 8: Concat (Bahdanau) Attention Layer.

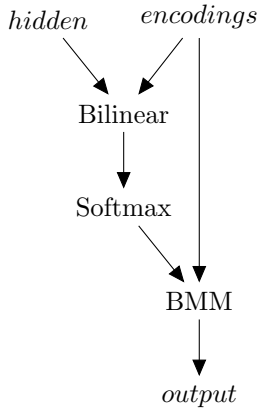

Figure 9: General Attention Layer.