NEURAL MACHINE TRANSLATION

A thesis submitted to
Loyola Marymount University
The Mathematics Department
in partial fulfillment of the requirements
for Graduation with the Bachelor of Science Degree

by

Quinn Lanners

May 2019

NEURAL MACHINE TRANSLATION

Senior Thesis by

Quinn Lanners

Dr. Thomas Laurent, Thesis Director

Neural Machine Translation is the primary algorithm used in industry to perform machine translation. This state-of-the-art algorithm is an application of deep learning in which massive datasets of translated sentences are used to train a model capable of translating between any two languages. The architecture behind neural machine translation is composed of two recurrent neural networks used together in tandem to create an Encoder Decoder structure. Attention mechanisms have recently been developed to further increase the accuracy of these models. In this senior thesis, the various parts of Neural Machine Translation are explored towards the eventual creation of a tutorial on the topic. In the first half of this paper, each of the aspects that go into creating a NMT model are explained in depth. With an understanding of the mechanics of NMT, the second portion of this paper briefly outlines enhancements that were made to the PyTorch tutorial on NMT to create an updated and more effective tutorial on the topic.

Thesis written by

Quinn Lanners

Approved by

Dr. Thomas Laurent, Thesis Director Date

Dr. Patrick Shanahan, Mathematics Department Chair Date

Contents

CHAPTER 1

Introduction

As technology continues to advance, the world becomes more and more connected. Physical distance is no longer as big of a roadblock as it once was; allowing people to connect with one another over a number of different social platforms. Yet, while distance may not divide us like it once did, the vast number of spoken languages continues to make universal communication difficult. With over 30 languages with 50 million or more speakers in the world,[1] language barriers are a common occurrence that hinder communication and collaboration all across the globe. And while translators have been effectively overcoming these barriers for centuries,[2] the demand for translation far outweighs the supply of translators available. However, just as it helped overcome the roadblock of physical distance, technology is continually improving our ability to overcome language barriers as well. Currently, the most effective manner by which technology is able to accomplish this task is through neural machine translation (NMT). In fact, NMT is the algorithm behind the globally utilized Google Translate system.

While NMT is a relatively new concept, the idea of machine translation has been studied on-and-off for several decades. Warren Weaver was the first to propose the idea of machine translation in 1949.[3] Weaver was inspired by the concepts of code breaking from WWII and the idea of an underlying similarity between all languages. Since the conception of machine translation in 1949, research on the topic has transitioned through several active and stagnant periods. Before NMT, statistical machine translation (SMT) provided the most state-of-the-art results. While many initially believed that SMT would eventually become the answer to machine translation, several issues, including the number of components that went into a single translation model and the lack of generalizability of a model, stagnated SMT progress and prevented SMT from ever providing perfect translations.

After multiple decades of SMT research, in the past five years NMT has arose as a far superior method at performing the task of machine translation. The rise of neural machine translation is due in large part to the increasing popularity of deep learning and the advanced power of GPUs and computers. Cho et. al was the first group to propose the concept of NMT in their paper ”Learning Phrase Representations using RNN Encoder-Decoder for Statistical

[1] Ethnologue Languages of the World, ed. *Summary by language size*. `https://www.ethnologue.com/statistics/size`. [Online; accessed 2-April-2019]. 2019.

[2] Jean Delisle and Judith Woodsworth. *Translators through History: Revised edition*. Vol. 101. John Benjamins Publishing, 2012.

[3] Fatemeh Nahas. “A Brief Review of Neural Machine Translation (NMT)”. in: *SAMPLE TRANSLATIONS* (), p. 15.

Machine Translation".[4] In the paper, Cho et. al proposed the use of two separate RNN networks to perform machine translation, a structure they called an RNN Encoder-Decoder. While the paper initially garnered little attention, it eventually led to a revolution in machine translation. The power of NMT was first introduced to the public at the 2015 OpenMT machine translation competition, where the model was so successful that over 90% of teams utilized the NMT structure in the following year's 2016 OpenMT competition.[5] Now, five years since its conception, NMT has taken off, capturing the attention of tech giants like Google and Facebook as the cutting edge technology in machine translation. This paper will investigate the mechanics and mathematical foundations of the NMT Encoder-Decoder architecture. Furthermore, this paper will look into the recent trend of attention-mechanisms and the added benefit they provide to NMT. Finally, the research portion of this paper will critique and enhance the current neural machine translation tutorial on the PyTorch website. PyTorch is one of the two most widely used machine learning libraries in Python (with TensorFlow being the other). And while the current PyTorch tutorial on machine translation does an adequate job of building a tutorial level translation model, there are several areas for improvement in both the explaining of the topics and the actual implementation of the code.

[4]Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[5]Ondřej Bojar et al. "Findings of the 2016 conference on machine translation". In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Vol. 2. 2016, pp. 131–198.

CHAPTER 2

Background and Review

The RNN Encoder-Decoder structure is composed of two separate RNN networks: the encoder and the decoder. Towards understanding how this architecture works, it is first essential to understand the basics behind neural networks and recurrent neural networks (RNN). Therefore, this background section will first cover the fundamentals of a neural network and the key distinctions that make a neural network a recurrent neural network. From there, this section will take a brief look at the data processing technique of One Hot Encoding before finally diving into the Encoder-Decoder structure. Lastly, this section will discuss Long-Short Term Memory RNNs and Attention mechanisms; two concepts which vastly improve the effectiveness of NMT.

2.1. Neural Network

The objective of any neural network is to train a set of learnable parameters to perform prediction tasks on inputted data. Any dataset used to train a model is made up of input data x composed of L_x features and an output target, y. In simpler machine learning tasks, this output target, y, is a single number corresponding to a value or class. However, in more complicated machine learning tasks, such as machine translation, this output target, y, is composed of L_y features. Before training begins, preprocessing of the data is performed. In general, for any machine learning task using a neural network the steps are as follows:

(1) Any preprocessing steps needed to make non-numeric data interpretable by computers are performed (see **One Hot Encoding**).
(2) The data is split into $Train$, $Validation$, and $Test$ sets. A typical split is around 70%:20%:10%.
(3) Train data is split into mini-batches for training purposes.

With the data prepared, the training process can begin. At the most basic level, the process of training a machine learning model is made up of 4 separate steps. When working with a train set of size n, the first step of a neural network is to take an input x_i for $i \in [1, n]$ and predict a probability vector, $\hat{y}_i$, in the following way:

$$\hat{y}_i = \text{softmax}(Wx_i) \tag{1}$$

The softmax function in (1) is defined as follows: If $\vec{v}$ is a vector in $\mathbb{R}^n$, $\vec{p}$ = softmax($\vec{v}$) is the vector in $\mathbb{R}^n$ defined as:

$$\vec{p} = \text{softmax}(\vec{v}) = \begin{bmatrix} \frac{e^{v_1}}{\sum_j e^{v_j}} \\ \vdots \\ \frac{e^{v_N}}{\sum_j e^{v_j}} \end{bmatrix} \tag{2}$$

In (1), W is referred to as a weight matrix, and is composed of a number of learnable parameters. Since the objective of W is to predict y_i for x_i, the dimensions of W in the above example are (L_y, L_x). In this way, the matrix-vector multiplication Wx_i transforms the input vector of length L_x (the number of input features) into a vector of length L_y (the number of classes).

With these predicted classes, the second step in the training process is to compute a loss l_i from the computed $\hat{y}_i$ as shown below:

$$l_i = LossFunction(\hat{y}_i, y_i) \tag{3}$$

The $LossFunction$ can be a number of different functions that analyze the accuracy of the predicted $\hat{y}$. Now, the third step of the training process is to repeat steps (1) and (2) on x_i for every $i \in [1, n]$ and take a summation over l_i to obtain a total loss L for a single loop through the entire dataset. This process is depicted by the following equation:

$$L = \sum_{i=1}^{n} l_i \tag{4}$$

A complete run through every training point in the dataset is referred to as a single epoch.

Now, we can use this L to update W, the final step of the training process. L is differentiated with respect to W to obtain $\partial L/\partial W$. Since L is being differentiated by W, $\partial L/\partial W$ will retrieve a different gradient for each value in W, thus, $\partial L/\partial W$ will be a matrix the same size as W with each index corresponding to the gradient of the loss with respect to the weight in W at that particular index. Therefore, we can use $\partial L/\partial W$ to update W as follows:

$$W = W - lr(\partial L/\partial W) \tag{5}$$

lr is a hyperparameter referred to as the learning rate and determines by how much to update W.

Once complete, these four steps are performed iteratively for a specified number of epochs to train the model. These four steps which make up the process of training the network are depicted in Figure 1.

In order to track the progress of the training process, the accuracy of the model on the validation set is computed at predetermined points in the training (ex. every 5 training loops). This is done by computing the loss on the validation set, L_{val} exactly like in the training steps 1-3. Thus, the only difference between this process and the training process performed on the train set is that step 4 of the training process, where the parameters of the weight matrices are updated, is not performed. By calculating this loss, L_{val}, at points throughout training, the ability for the model to accurately make predictions on data beyond the train set can be tracked. This step is crucial in avoiding issues such as overfitting. Finally, once the training process is complete, the loss is computed on the test set, L_{test}, as another way of determining the effectiveness of

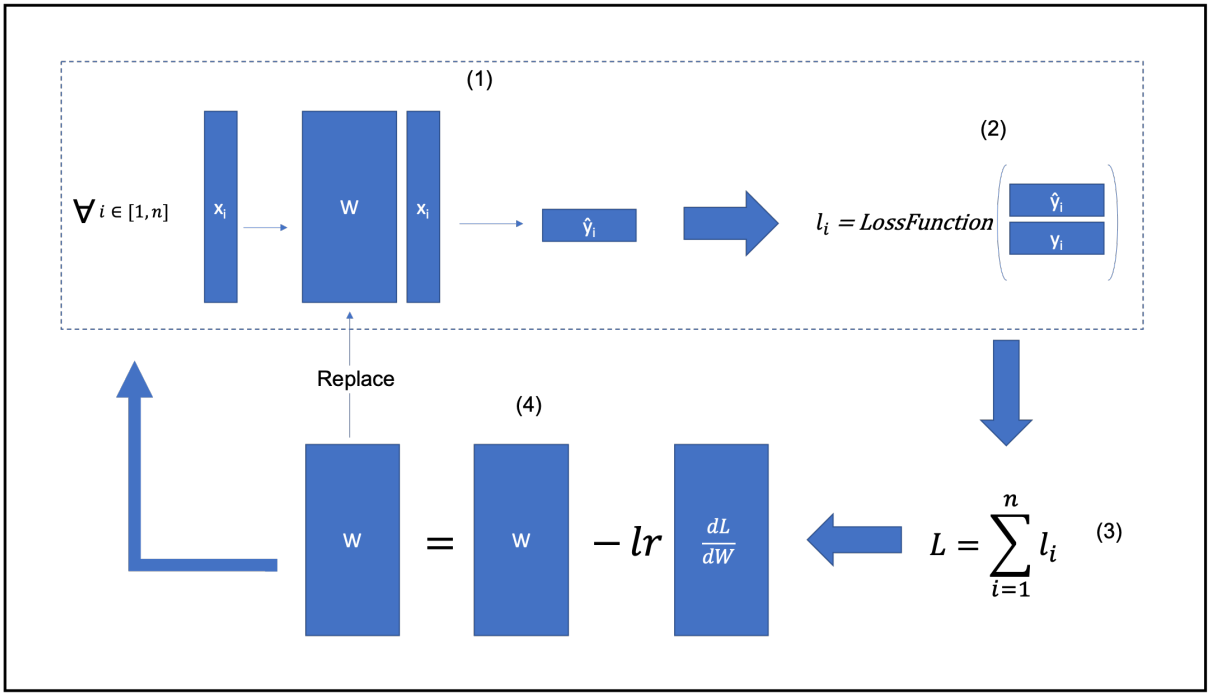

FIGURE 1. The training process for a one-layer neural network.

the model. While the loss on the validation set, L_{val}, should represent the model's ability to generalize to data outside of the training set, the test set is used at the completion of the entire training process as a way of exposing the model to data it has never before seen and thus double checking this characteristic.

While it is generally good practice to use both a validation set and a test set, oftentimes a validation set is not used. This allows for more data to be used in the train set (for example an 80%:20% train set:test set split is common). If this method is employed, the accuracy of the model is just computed on the test set, rather than the validation set, at predetermined points in training. And while this does slightly decrease the ability to determine how the model generalizes to outside data, it oftentimes is the superior method due to the increased training set size.

As an important note, in the above example, since there was only a single weight matrix W, this type of network is referred to as a one-layer neural network. However, in the application of deep learning, several of these weight matrices are applied to the input data, with non-linear functions between them, before a prediction vector is calculated. While this paper will not cover this topic in detail, it is important to mention as it is the standard in neural networks today.

2.2. Recurrent Neural Network

In the task of neural machine translation, the input data comes in the form of time-series data. Recurrent Neural Networks are a particular type of neural network designed specifically

to deal with the intricacies of temporal data. In the case of time-series data,[1] the input of each training point, x_i, consists of a number of sequential data points $x_{i_{t=1}}, x_{i_{t=2}}, ..., x_{i_{t=L_x}}$ of equal size, where L_x is the length of the time-series. Just like a standard neural network, the aim of a RNN is to predict the label y_i for each x_i. Training a RNN is very similar to training a standard neural network, with just slight modifications to steps 1 and 4 outlined in the **Neural Network** section above.[2] In step 1 of training, a RNN analyzes each data point $x_{i_{t=1}}, x_{i_{t=2}}, ..., x_{i_{t=L_x}}$ in consecutive order and stores information in a hidden vector, h_t, to ultimately make a prediction, $\hat{y}_i$, at the end of the sequence. $h_{t=0}$ is initialized to a zero vector, and updated at each time step t. This update process is defined by the following equation:

$$h_t = \sigma(Rh_{t-1} + Wx_{i_t}) \tag{6}$$

In (6), R and W are weight matrices and σ is a non-linear function such as tanh or sigmoid. Using this architecture, the hidden vector h_t captures the important information from the time series data by being updating at each time step t. Thus, at any time step $s < L_x$, the hidden state $h_{t=s}$ contains information about the data points $x_{i_{t=1}}$ up to $x_{i_{t=s}}$. In this way, at any time step s, the hidden vector, $h_{t=s}$, can be used to create a prediction, $\hat{y}_i$. This prediction vector is created just as in the first step of training a standard neural network, and is shown below:

$$\hat{y}_i = \text{softmax}(Uh_{t=s}) \tag{7}$$

Where, in (7), U is another weight matrix which converts the size of h_s to the size of the possible outcome classes. The modified first training step for a RNN is shown in Figure 2. Now, using this $\hat{y}_i$, steps 2 and 3 of training a RNN are exactly the same as training a standard neural network in that (2) the loss is computed for $\hat{y}_i$ and (3) steps 1 and 2 are repeated for each training example and the losses are summed together to obtain a total loss L. With this L, a slightly modified step 4 is conducted to update all of the weight matrices W, R, and U. In particular, L is differentiated with respect to each weight matrix and used to update each matrix as depicted in the following set of equations:

$$\begin{aligned} W &= W - lr(\partial L/\partial W) \\ R &= R - lr(\partial L/\partial R) \\ U &= U - lr(\partial L/\partial U) \end{aligned} \tag{8}$$

Just as in any neural network, this process is then repeated for multiple epochs in order to train the RNN model. With an understanding of the structure of a RNN, let's turn our attention towards the topic of neural machine translation. However, we must first address how we transform text data into computationally friendly data.

2.3. One Hot Encoding

In machine translation, we are given a sentence in one language and tasked with translating the sentence to another language. The most obvious roadblock with machine translation is the

[1] David E Rumelhart et al. "Sequential thought processes in PDP models". In: *Parallel distributed processing: explorations in the microstructures of cognition* 2 (1986), pp. 3–57.

[2] Minh-Thang Luong. "Neural Machine Translation". In: (2016). (Unpublished doctoral dissertation) and [Online; accessed 27-March-2019].

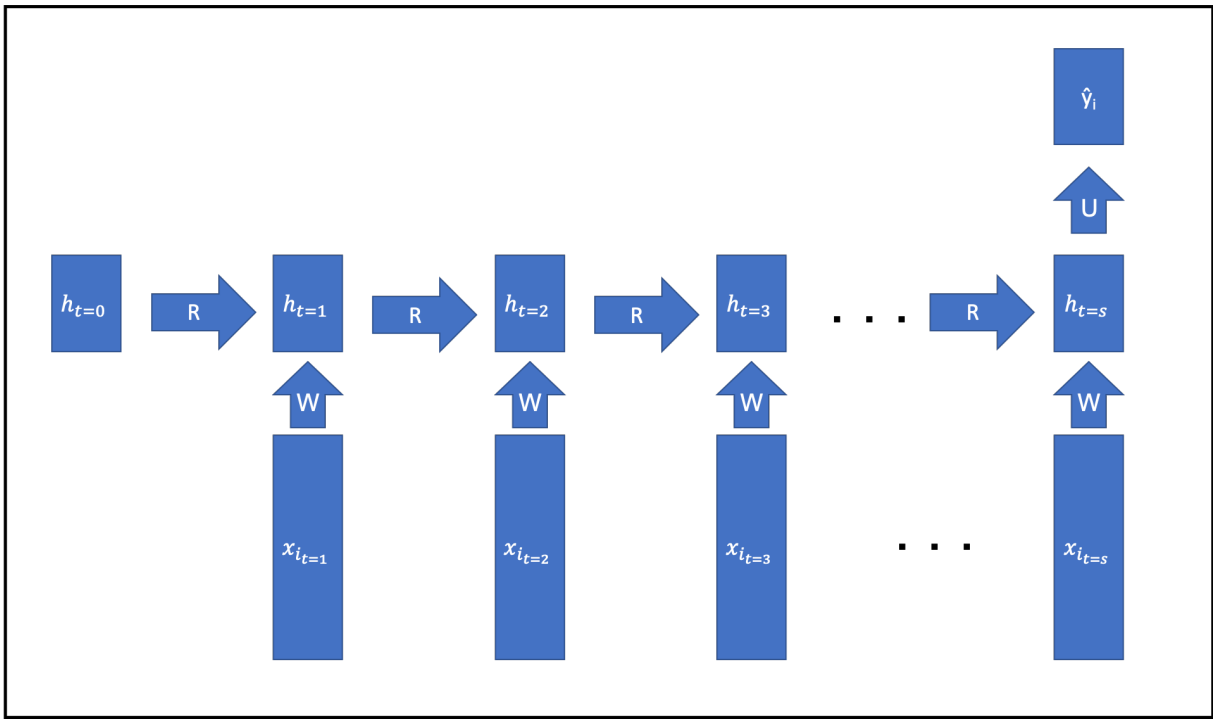

FIGURE 2. Predicting a label using a RNN.

issue of performing machine learning with textual data. The answer to this issue is one hot encoding, which involves converting each word in the dataset into a vector with a 0 at every index with the exception of a single 1 at the index corresponding to that particular word.[3] To determine the size of these one hot encoding vectors, a separate vocabulary is created for both the input and output languages. Ideally, these vocabularies would be every unique word in each of these respective languages. However, given that a single language can have millions of unique words, vocabularies often consist of a subset of the most common words in each language. Typically, these vocabularies are allowed to be in the hundreds of thousands (if not millions) of words for each language. But, for the sake of example, let's let the vocabulary for our input language (English) consists of the words in Table 1. We are then able to use this vocabulary to make one-hot encoding vectors for each of the word in the input language where each vector will be the same size as the input vocabulary. For example, if one of our input sentences is *"the blue whale ate the red fish"* the one hot encoding vectors for this sentence are shown in and Figure 3

The start of sentence, <SOS>, and end of sentence, <EOS>, tags are added to the vocabulary to denote the start and end-points of sentences. These tags become important during the training process, which will become apparent in the next section. And, while not shown here, this same process must be done to the output sentences of the dataset (i.e. the ys).

[3]Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation", op. cit.

TABLE 1. Input Language Vocabulary

a	0
the	1
red	2
orange	3
blue	4
black	5
fish	6
whale	7
beaver	8
ate	9
drank	10
<SOS>	11
<EOS>	12

$$the = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \; blue = \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \; whale = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix} \; ate = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix} \; the = \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \; red = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \; fish = \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

FIGURE 3. One hot encoding of *"the blue whale ate the red fish."*

2.4. Encoder-Decoder

With a way to transform textual data into numerical vectors, we can now begin the task of NMT. The training set for an RNN translating from language A to language B is composed of pairs of an input sentence in language A (the x_i) and target sentence in language B (the y_i). The structure of a single training pair is shown below:

$$\begin{aligned} x_i &= x_{i_{t=1}}, x_{i_{t=2}}, ..., x_{i_{t=L_x}} \mid L_x = length(input\ sentence) \\ y_i &= y_{i_{t=1}}, y_{i_{t=2}}, ..., y_{i_{t=L_y}} \mid L_y = length(output\ sentence) \end{aligned} \tag{9}$$

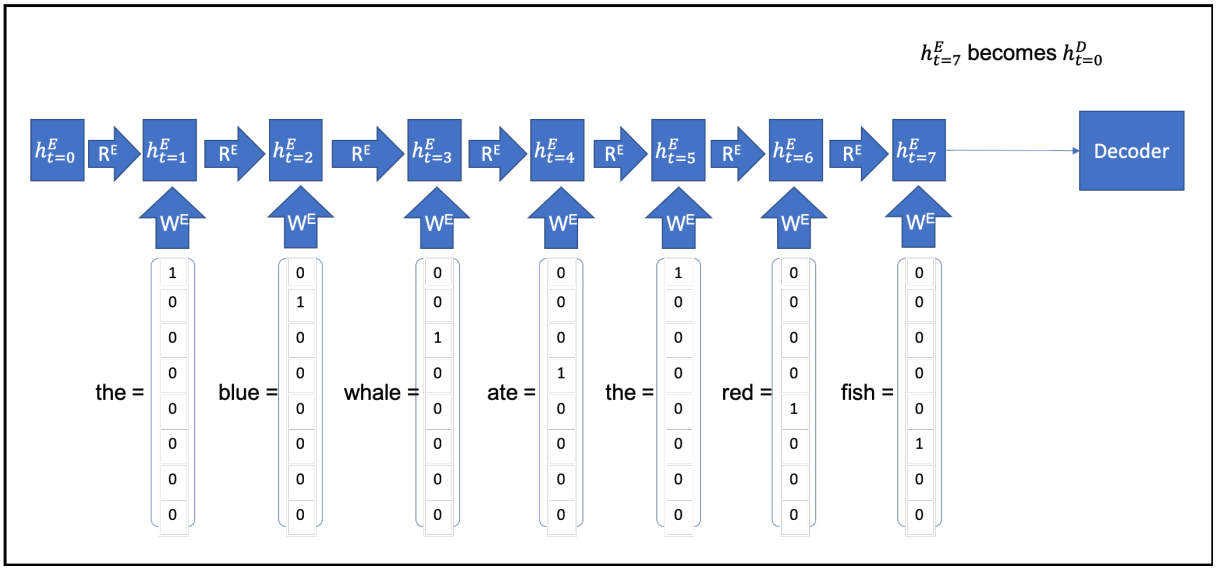

FIGURE 4. Encoder portion of NMT

The obvious difference between this dataset and the type used in the RNN section is the fact that the target, y_i, is a variable length vector rather than a single label. In order to combat this issue, NMT utilizes the Encoder-Decoder architecture.[4] In this structure, a first RNN (the encoder) analyzes the input sentence and passes its final hidden state, $h^E_{t=L_x}$, onto a second RNN (the decoder) to use as its first hidden state, $h^D_{t=0}$. For example, if we wanted to translate the English sentence *"the blue whale ate the red fish"* to the French sentence *"la baleine bleue a mangé le poisson rouge"*, the encoder portion of this translation would take *"the blue whale ate the red fish"* as input, and feed its final hidden vector, $h^E_{t=7}$ ($t = 7$ because the sentence is 7 words long), to the decoder to use as its first hidden vector $h^D_{t=0}$ (Figure 4). On the decoder portion of the Encoder-Decoder, a separate RNN predicts words for the variable length output statement $\hat{y}_i$. In the first time step, the $h^D_{t=0}$, retrieved from the encoder, and the <SOS> token, which is used as the first input vector $x_{i_{t=1}}$, are used to compute $h^D_{t=1}$. However, unlike the vanilla RNN described in the RNN section, the decoder outputs a predicted word for each time step t up to L_y. Thus, the Decoder uses the hidden state $h^D_{t=1}$ to compute a predicted word $\hat{y}_{i_{t=1}}$ in the same manner as the final step of a vanilla RNN. This process is depicted in the following equation and in Figure 5.

$$\hat{y}_{i_{t=1}} = \text{softmax}(U^D h^D_{t=1}) \tag{10}$$

With this prediction vector, $\hat{y}_{i_{t=1}}$, a loss is computed for the first word, where $y_{i_{t=1}}$ is the correct word for that output sentence at $t = 1$. The equation to compute the loss is shown below:

$$l_i = LossFunction(\hat{y}_{i_{t=1}}, y_{i_{t=1}}) \tag{11}$$

When testing the model, the index with the highest value in the prediction vector, $\hat{y}_{i_{t=1}}$, determines what word becomes the next input vector, $x_{i_{t=2}}$. However, during training, a more

[4] Ibid.

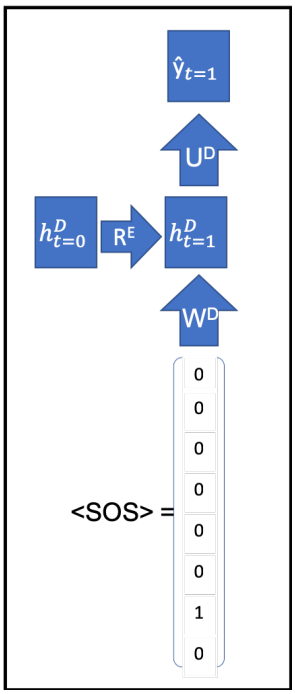

FIGURE 5. First step of Decoder

common procedure, referred to as *Teacher Forcing*, is used to speed up the training process. In teacher forcing, rather than using the predicted word, the Decoder simply inputs the correct word, $y_{i_{t=1}}$, as the next input vector, $x_{i_{t=2}}$. This process is then carried out L_y times until a prediction vector has been created for each word in the output sentence (Figure 6).

The losses for each word are summed to obtain a loss for the whole sentence, and this process is repeated for each word in the training set to obtain a total loss L. From here, L is differentiated with respect to each weight matrix and used to update both the encoder's and the decoder's weights, as shown in following set of equations:

$$\begin{aligned} W^E &= W^E - lr(\partial L/\partial W^E) \\ R^E &= R^E - lr(\partial L/\partial R^E) \\ W^D &= W^D - lr(\partial L/\partial W^D) \\ R^D &= R^D - lr(\partial L/\partial R^D) \\ U^D &= U^D - lr(\partial L/\partial U^D) \end{aligned} \tag{12}$$

Now, with an understanding of NMT, let's take a look at a more sophisticated RNN architecture which will go a long way in improving the results of machine translation.

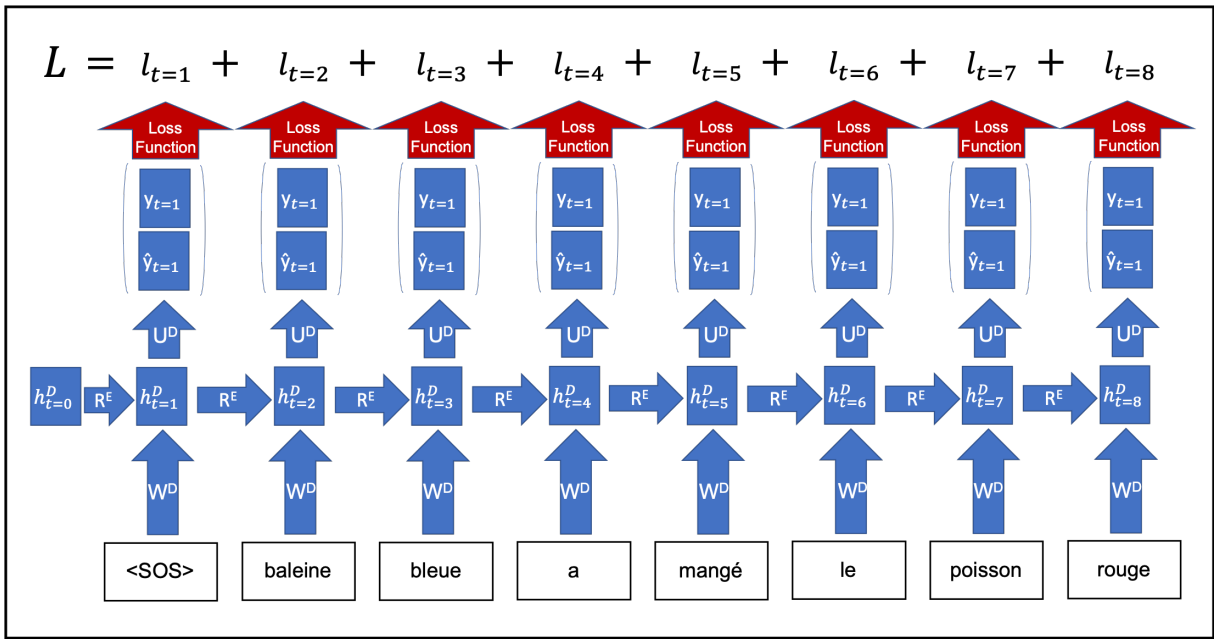

FIGURE 6. Decoder portion of NMT

2.5. Long Short-Term Memory

The Long Short-Term Memory (LSTM) architecture is perhaps the most widely used RNN structure in large part due to its effectiveness at dealing with the issue of vanishing gradients.[5] LSTM is modeled after the traditional RNN format, but incorporates a number of "gates," which work together with a memory cell, to smooth the back-propagation procedure and create a seamless flow through the network. In particular, at any time step $t = s$ the LSTM structure updates the hidden state, $h_{t=s}$, by incorporating information from the current input vector, $x_{t=s}$, the previous hidden state, $h_{t=s-1}$, and an additional memory cell, $c_{t=s}$. In order to conduct this update, three separate gates, known as the input gate (i), forget gate (f), and the output gate (o), are used to determine how much information to draw from the current input versus how much information to draw from the memory cell. All three of these gates are calculated using information from the current input vector, $x_{t=s}$, and the previous hidden state, $h_{t=s-1}$. To use information from each of these vectors, they are concatenated together $[h_{t=s-1}, x_{t=s}]$; meaning the $x_{t=s}$ vector of size Xx1 is added to the end of the $h_{t=s-1}$ vector of size Hx1 to create a vector of size $(H + X)$x1. A separate weight matrix for each gate is then used to transform this concatenated vector, $[h_{t=s-1}, x_{t=s}]$, into a vector the same size as the memory cell, and each of these transformed vectors are then passed through a σ function to produce a vector with

[5]Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*. 2012.

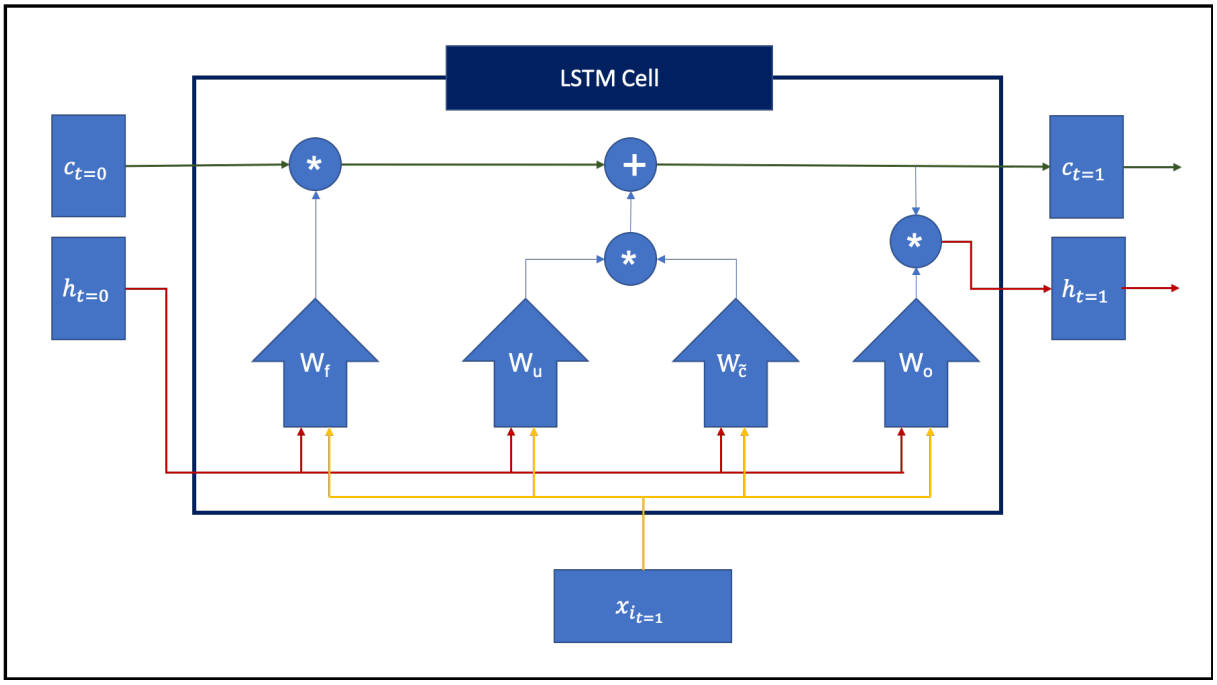

FIGURE 7. First step of LSTM

weighted indices that sum to one. The equations that define each of these gates are below:

$$\begin{aligned} i &= \sigma(W_i[h_{t=s-1}, x_{t=s}]) \\ f &= \sigma(W_f[h_{t=s-1}, x_{t=s}]) \\ o &= \sigma(W_o[h_{t=s-1}, x_{t=s}]) \end{aligned} \tag{13}$$

Before these gates can be used, the current input vector $x_{t=s}$ and the previous hidden state, $h_{t=s-1}$ are used to generate $\tilde{c}_{t=s}$, which is defined by the following equation:

$$\tilde{c}_{t=s} = tanh(W_{\tilde{c}}[h_{t=s-1}, x_{t=s}]) \tag{14}$$

Now, the gates and $\tilde{c}$ are used to calculate a new $c_{t=s}$ which is subsequently used with $c_{t=s-1}$ to generate a hidden state $h_{t=s}$. This process is depicted below:

$$\begin{aligned} c_{t=s} &= (i * \tilde{c}_{t=s}) + (f * c_{t=s-1}) \\ h_{t=s} &= o * tanh(c_{t=s}) \end{aligned} \tag{15}$$

In (15), the $*$ symbol signifies element wise multiplication; meaning the corresponding indices of each of the vectors are multiplied together. Figure 7 puts all these steps together into a visualization of the first step of an LSTM.

Replacing this LSTM architecture with the vanilla RNN in the NMT section above, the full Encoder can be visualized as in Figure 8. This same substitution is performed for the Decoder RNN to convert an NMT model into a full LSTM NMT architecture. The other steps of training remain exactly the same, with predictions computed from the hidden vector, h_t, and the total loss, L, determined from these predictions. The total loss is then differentiated with respect

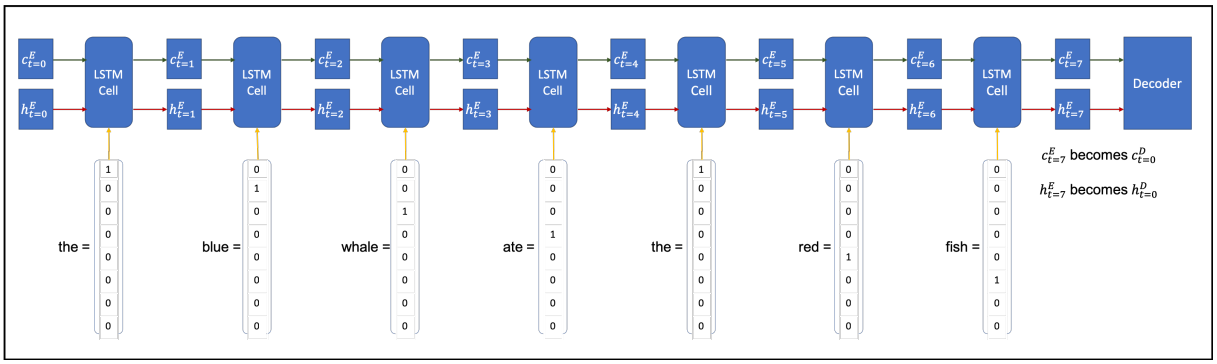

FIGURE 8. Encoder of NMT with LSTM

to each of the weight matrices in the encoder and the decoder and updated as shown in the following set of equations:

$$\begin{aligned}
W_i^E &= W_i^E - lr(\partial L/\partial W_i^E) \\
W_f^E &= W_f^E - lr(\partial L/\partial W_f^E) \\
W_o^E &= W_o^E - lr(\partial L/\partial W_o^E) \\
W_{\tilde{c}}^E &= W_{\tilde{c}}^E - lr(\partial L/\partial W_{\tilde{c}}^E) \\
W_i[^D &= W_i^D - lr(\partial L/\partial W_i^D) \\
W_f^D &= W_f^D - lr(\partial L/\partial W_f^D) \\
W_o^D &= W_o^D - lr(\partial L/\partial W_o^D) \\
W_{\tilde{c}}^D &= W_{\tilde{c}}^D - lr(\partial L/\partial W_{\tilde{c}}^D) \\
U^D &= U^D - lr(\partial L/\partial U^D)
\end{aligned} \tag{16}$$

Along with creating a more seamless flow back through the network for back-propagation, by creating the h_t at each time-step t using both information from the input vector, x_t, and the memory cell state from the previous time step, c_{t-1}, LSTMs also helps preserve long term dependencies that often exist in time series data. This is a highly desirable trait of a MT model given that oftentimes in the task of machine translation a word towards the end of the sentence may be highly dependent on a word towards the beginning of the sentence. It is because of these numerous advantages that the LSTM architecture was chosen to perform the task of NMT in this research.

2.6. Attention Mechanism

Perhaps the most difficult aspect of machine translation is handling longer pieces of text. Given the time-step nature of the RNN structures used in machine translation, the longer the input sentence, the more steps needed by an encoder to create a thought vector, $h^E_{t=L_x}$, to be inputted into a decoder . Furthermore, the longer the input sentence becomes, the more information that needs to be stored in this thought vector and then subsequently decoded by the decoder. While a sentence as short as shown below would probably not cause major problems for a simple machine translation architecture, for the sake of example let's consider translating the following sentence from English to French:

The black cat ran across the pond. >> Le chat noir a traversé l'étang.

Observe how in this sentence, similar to most sentences from English to French, the words towards the beginning of the English sentence seem to correspond well with the words towards the beginning of the French sentence (the=le, black=noir, cat=chat, etc.). Thus, it can be understood how when the sentence becomes longer, it may be difficult for the encoder to preserve information from the beginning of the sentence in the thought vector it produces and passes onto the decoder. Subsequently, the decoder is provided with little information on how it should start its outputted sentence. In order to combat this issue, it had previously been found beneficial to reverse the order of the input sentence in order to move the words thats correspond to the first words of the outputted sentence closer to the end of the encoder; ensuring that information on these first few words becomes encoded in the thought vector and passed onto the decoder.[6] However, while this method has minimal success with translating between some languages, it does not generalize well to translation between all languages (given that the sentence structure of languages varies greatly). Furthermore, this method sacrifices quite a bit of the information from the end of the input sentence, as that information is inputted first into the encoder and is oftentimes not accessed again until the end of the output sentence, creating several steps over which this information often becomes lost.

A more full-proof fix to the issue of translating long pieces of text, the attention mechanism proposed by Bahdanau in 2014,[7] has become the standard in machine translation attention architecture. In general, Luong's attention mechanism utilizes the hidden states outputted by the encoder at each time step, rather than just the final thought vector. In doing so, the decoder can "refer back" to the source sentence at each step to help determine which parts of the inputted sentence are most helpful in determining what the next outputted word should be. In general, the attention mechanism gives heavier weights to encoder hidden states which more closely resemble the current hidden state.This is done by computing a context vector d_t, at each time step t in the decoder, as a weighted average over a selection of the hidden states outputted from the encoder. This weighted average is computed by first creating an alignment vector, a_t, whose length corresponds to the number of source hidden states the weighted average is being taken over. This a_t is computed by comparing the similarity between the current hidden state, h_t^D, and each separate encoder hidden state, $h_{t'}^E$, being considered. In this way, a_t is defined by the following equation:

$$a_t[s] = \frac{exp(score(h_t^D, h_{t'=s}^E))}{\sum_{\tau=0}^{L_x} exp(score(h_t^D, h_{t'=\tau}^E))} \quad \forall s \in [0, L_x] \tag{17}$$

Where h_t^D is the current hidden state being inputted into the decoder, $h_{t'=\tau}^E$ is an encoder hidden state for some time-step τ, L_x is the number of encoder hidden states which are being weighted over, and exp is the exponential function. By computing an element $a_t[s]$ $\forall s \in [0, L_x]$, a_t becomes a vector of size L_xx1. Finally, in (17), $score$ is the function which computes the

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

similarity between the hidden states, and can be one of a variety of different forms:

$$\begin{aligned} score(h_t^D, h_{t*}^E) &= (h_t^D)^T(h_\tau^E) && (dot) \\ score(h_t^D, h_{t*}^E) &= (h_t^D)^T(W_a h_\tau^E) && (general) \\ score(h_t^D, h_{t*}^E) &= V_a^T tanh(W_a[h_t^D : h_\tau^E]) && (concatenation) \end{aligned} \tag{18}$$

In the $general$ and $concatenation$ score functions, W_a and V_a are learnable weight matrices.

Using these attention scores, Luong proposed two attention mechanisms; global and local. The global approach is the simpler form of attention and the one which will be covered in this paper. For details on local attention refer to (Luong et al., 2015).[8]

Global attention considers all parts of the input sequence, using information from each of the hidden states in the encoder to help predict each of the target words. In order to use information from every encoder hidden state, the softmax function is used to take a weighted average over all of the indices in a_t to create a vector, a'_t, whose indices sum to one. In this way, each index $a_t[s]$ where $s \in [0, L_x]$ corresponds to the weight of that encoder hidden state, $h_{t=s}^E$. Thus, the context vector, d_t, is simply the weighted average over all the source hidden state's $h_{t'=s}^E \; \forall s \in [0, L_x]$ and is computed as follows:

$$\begin{aligned} a'_t &= \text{softmax}(a_t) \\ d_t &= \textstyle\sum_{s=0}^{L_x} a'_t[s] * h_{t'=s}^E \end{aligned} \tag{19}$$

This context vector d_t is then used along with the the decoder hidden state, h_t^D, to compute $\tilde{h}_t^D$, which takes the place of h_t^D as the outputted hidden state. This process is depicted by the following equation:

$$\tilde{h}_t^D = tanh(W_{\tilde{h}}[d_t : h_t^D]) \tag{20}$$

From here the prediction vector, $\hat{y}_t$, is computed by simply replacing the h_t^D with the new $\tilde{h}_t^D$, as shown below:

$$\hat{y}_i = \text{softmax}(U\tilde{h}_t^D) \tag{21}$$

$\tilde{h}_t^D$ also replaces h_t^D as the hidden state fed to the next step step $t+1$ in the RNN. The attention mechanism in the first step of the decoder is depicted in Figure 9.

[8]Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

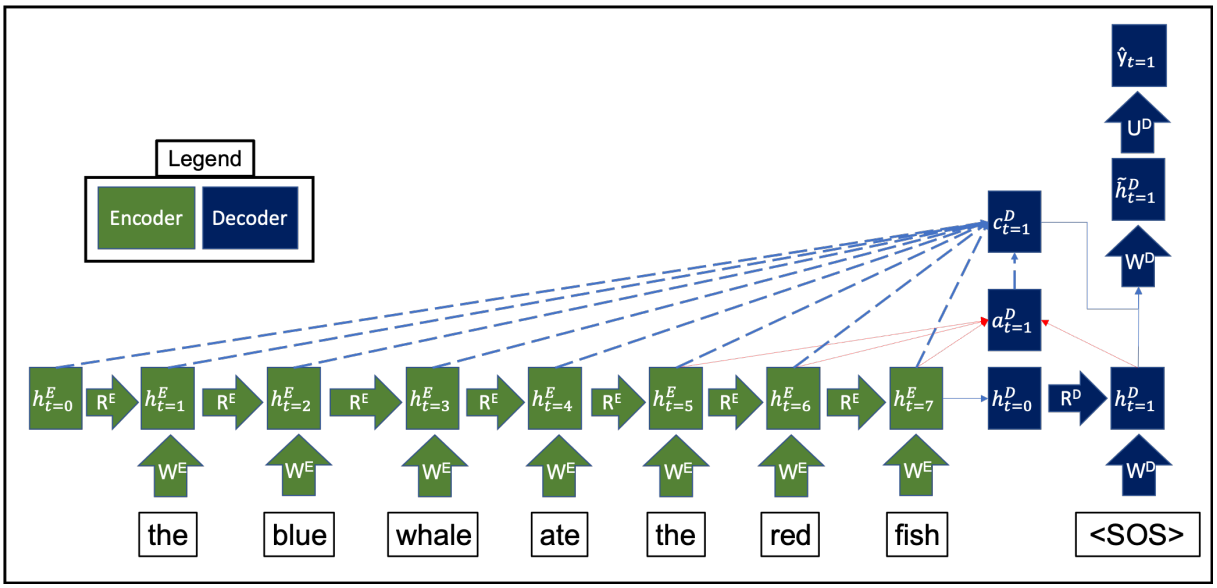

FIGURE 9. Step 1 of Decoder using Global Attention

CHAPTER 3

Experimental Design

In order to apply the above concepts, this research took on the task of enhancing the PyTorch tutorial on neural machine translation titled "Translation with a Sequence to Sequence Network and Attention".[1] While this tutorial employs an encoder-decoder structure and utilizes an attention mechanism, there are a number of faults in the implementation of this code. Primarily, while this code runs well on a CPU, it does not have the ability to harness the parallel computing power of a GPU. Particularly, there is no option to batchify the data in order to speed up the training process. Additionally, the PyTorch tutorial used a gated recurrent network, rather than the long-short term memory recurrent neural network discussed in this paper. While the performance difference between these two types of RNN architectures is often insignificant, recent findings suggests that LSTM models with large forget biases outperform GRU models.[2] Finally, and perhaps the most significant flaw of the tutorial, is that the model trained in the tutorial is never run on a test set. Thus, there is no way of knowing if the model has any ability to generalize to data outside of the test data.

In this project, the above enhancements were made to the PyTorch tutorial on machine translation. The following sections outline the dataset used, the model specifications, and the results of these enhanced models.

3.1. Dataset

The dataset used in this experiment was the same dataset employed by the PyTorch tutorial itself. This dataset was compiled by *Tatoeba* and consists of 167,130 sentence pairs of French and English. While not nearly as large as the datasets used to train state of the art models, this dataset provides a sufficient amount of sentence pairs to train a neural machine translation model with limited capabilities. In the case of the translation task at hand, we worked to generate a model that translated from French to English.

3.2. Initial Experiment

As a way of ensuring that the PyTorch tutorial results could be reproduced with the proposed enhancements, the preliminary experiment of this study directly compared the results of the enhanced model with the original PyTorch tutorial model. A direct comparison of the two model

[1] Sean Robertson. *Translation with a Sequence to Sequence Network and Attention*. Ed. by PyTorch. `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`. [Online; accessed 17-April-2019]. 2017.

[2] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.

TABLE 1. Model Architectures Comparison

	PyTorch Model	**My Model**
Hidden Size	256	440
Bidirectional	False	True
Number of Layers	1	2
Dropout	0.1	0.2
Batch Size	1	32
Learning Rate	0.01	1

TABLE 2. Start of Sentence Filters

i am	i'm
he is	he's
she is	she's
you are	you're
we are	we're
they are	they're

architectures is outlined in Table 1. In order to replicate the PyTorch tutorial, the enhanced model was trained on the same subset of the data as the tutorial model. This subset consisted of all sentences from the whole dataset that were ten words or less in length and the english sentence began with one of the word/words in Table 2. With these filters applied to the dataset, the set was trimmed to 10,853 sentence pairs of 2,926 unique French words and 4,490 unique English words. And, as referenced above, there was no test set used by the tutorial, and thus this subset was not split into a train and test pair; rather all pairs were used as a train set. Each of these models was trained using this smaller training set for 40 minutes on a GPU, and the cross entropy loss of each model at various points of the training process was collected. These losses are depicted in Figure 1. From this graph it is apparent that the enhanced model trained at a much faster rate, achieving a loss significantly lower than the PyTorch tutorial model in the 40 minute training period. The majority of this performance gain presumably came from the enhanced model's ability to utilize the parallel computing power of the GPU by training in batches, while the original PyTorch tutorial model still only trained with a batch size of one even on the GPU.

In a more qualitative sense, the superiority of the enhanced model can be seen in the translation of select sentence pairs from the training set. Table 3 below depicts this with two seperate sentence pairs from the training set. These examples work to show how in the short 40 minute training period, the enhanced model was able to develop a model more effective on translating sentences from the training set and thus a model that was fit better to the training data. Yet, despite this improvement, when attempting the translate sentences outside of the training set (especially sentences longer than ten words), the enhanced model performed quite poorly. For example, Table 4 shows an example translation of a sentence outside of the training set. (In

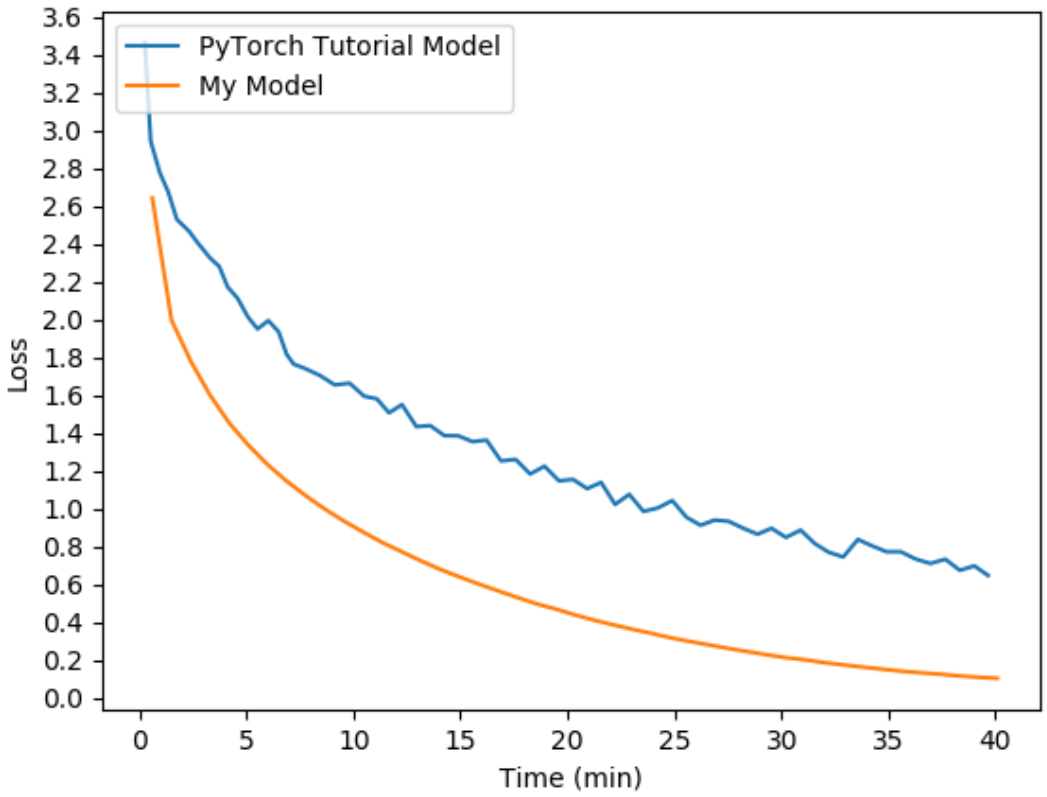

FIGURE 1. Cross Entropy Losses of PyTorch Model vs My Model during 40 Minute Training Period

TABLE 3. Sample Translations from Training Set

Input Sentence	je suis en bequilles pour un mois .	elles sont dingues de jazz .
Correct Target Sentence	i m on crutches for the next month .	they are crazy about jazz .
PyTorch Tutorial Model Translation	i m on business to go this . .	they re crazy about it .
My Model Translation	i m on crutches for the next month .	they are crazy about jazz .

Table 4, the last column of the **PyTorch Tutorial Model Translation** row is empty because the PyTorch tutorial model is unable to handle sentences longer than ten words in length.) Thus the second part of this experiment aimed to improve the enhanced model's ability to generalize to sentences outside of the dataset.

TABLE 4. Sample Translations of Sentences Outside the Train Set

Input Sentence	je suis en bequilles pour un mois .	j'ai fait très attention à ne pas réveiller le chien quand je suis parti tôt ce matin .
Correct Target Sentence	i hope you have an incredible day .	i was very careful to not wake the dog when i left early this morning .
PyTorch Tutorial Model Translation	i m not telling you some . .	
My Model Translation	i m looking for you another wrong .	i am very not around the morning .

TABLE 5. Sample Translations of Sentences Outside the Train Set for Model Trained on Larger Dataset

Input Sentence	j'ai fait très attention à ne pas réveiller le chien quand je suis parti tôt ce matin .
Correct Target Sentence	i was very careful to not wake the dog when i left early this morning .
Model Translation	i had a very careful not to wake up the dog when i left early this morning .

3.3. Further Experimentation

In the second portion of this study, slight modifications were made to the model and training hyper-parameters. In particular, the hidden size of the model was increased from 440 to 1080, the batch size was decreased from 32 to 10, and the learning rate was set initially to 0.5 and divided by five after every five epochs. However, more importantly, to create a more capable translation model the training set needed to be much larger. Thus, the start of sentence filters in Table 2 were removed and the dataset was trimmed to all sentences of length 40 words or less. With this less stringent criteria, the dataset was trimmed to 135,834 sentence pairs rather than 10,853. The max vocabulary was set to 20,000 words for each language resulting in a vocabulary size of 13,038 French words and 13,586 English words. Furthermore, in order to track the model's ability of generalizing to sentences outside the training set, these filtered sentence pairs were split into a train set consisting of 122,251 sentence pairs and a test set consisting of the remaining 13,583 sentence pairs. The model was trained for 15 full epochs through the training set and, now having a test set, the performance on the test set was computed after each training epoch. The cross entropy loss on both the train and test set after each epoch of the training process are shown below in Figure 2. With a significantly longer training period on a much larger training set, this model was able to achieve reasonable translation results on simple sentences outside the training set. Table 5 shows this model's relatively accurate translation of the same sentence that the previous model severely struggled with (Table 4). However, despite this improvement, the model is far from reaching anything close to state of the art translation results.

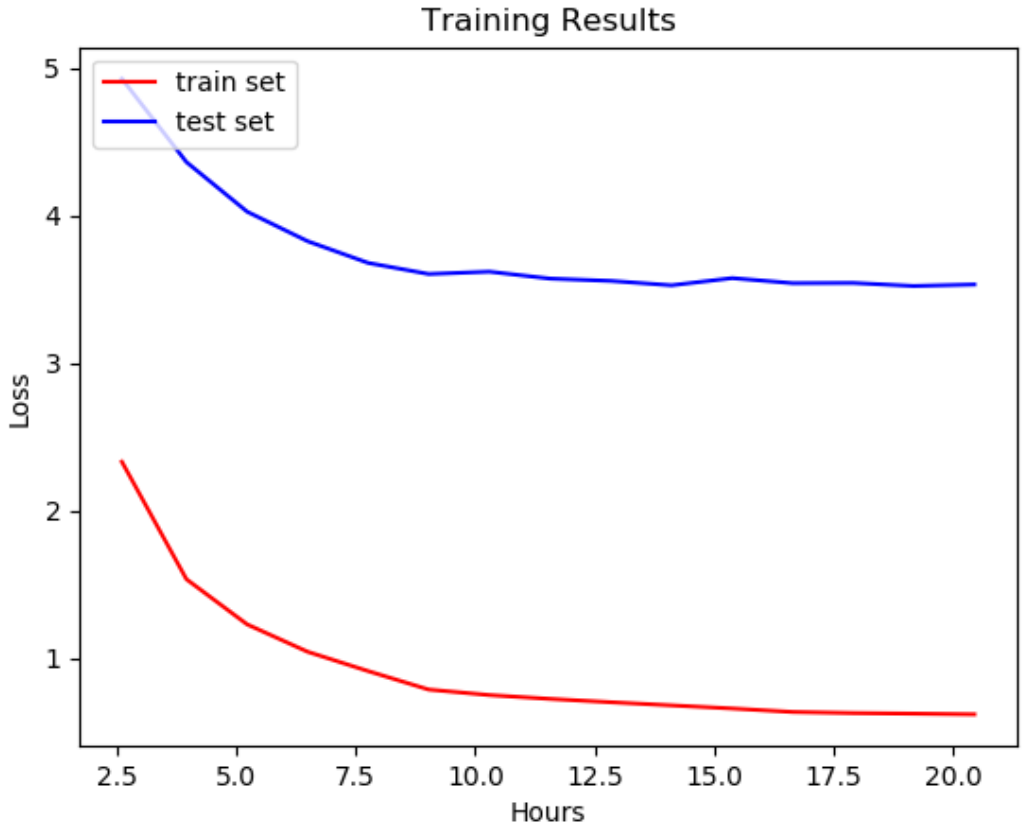

FIGURE 2. Cross Entropy Losses on Train and Test Sets over 15 Epochs of Training

3.4. Future Enhancements

This research focused primarily on understanding the Encoder Decoder structure and making improvements to the PyTorch tutorial on machine translation. Thus, the aim of this research was to create a model that was slightly more advanced and sophisticated than the current PyTorch tutorial model, but still capable of being easily reproduced by an individual looking to learn about neural machine translation. Moving forward, however, there are still several areas of improvements that can be made to the machine translation model outlined above. The easiest enhancement would be to simply train the existing model on a significantly larger dataset. Due to time constraints this was not performed, but would theoretically create an even more capable NMT model. More related to the architecture of the model, a more advanced attention mechanism could be implemented. In particular, global attention was used in this model, but research has found various local attention mechanisms to be much more effective.[3] Finally, while the cross entropy loss is one way of understanding a model's accuracy, there are several more

[3]Luong, Pham, and Manning, "Effective approaches to attention-based neural machine translation", op. cit.

sophisticated manners of quantifying a machine translation model's accuracy. In particular, a BLEU score algorithm could be implemented to more effectively understand the accuracy of the model's translations.

CHAPTER 4

Conclusion

With increased computational power, NMT has emerged as an effective method for machine translation. By utilizing the Encoder-Decoder structure, NMT is able to translate sentences with incredible accuracy. Furthermore, with the added power of attention mechanisms, machine translation is becoming even more powerful. The Python machine learning package, PyTorch, provides all of the necessary tools to create a NMT model. In fact, the PyTorch website itself has a tutorial on creating such a model. However, the model created in this tutorial is lagging in areas that are crucial to creating a model that fully utilizes the computational power of GPUs. Thus, this research aimed to enhance this NMT model, while keeping it at a tutorial scale. By increasing the size of the train set, adding a test set, batchifying the data, and tweaking a number of other model parameters, a much more effective machine translation model was ultimately created.

CHAPTER 5

Acknowledgements

I would like to give a special thank you to the Loyola Marymount University Mathematics Department and Dr. Thomas Laurent in particular. It was because of Dr. Laurent that I became interested in the topic of deep learning and he has been invaluable in my development as an academic and professional in this area of study.

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

Bojar, Ondřej et al. "Findings of the 2016 conference on machine translation". In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Vol. 2. 2016, pp. 131–198.

Cho, Kyunghyun et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

Delisle, Jean and Judith Woodsworth. *Translators through History: Revised edition*. Vol. 101. John Benjamins Publishing, 2012.

Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.

Luong, Minh-Thang. "Neural Machine Translation". In: (2016). (Unpublished doctoral dissertation) and [Online; accessed 27-March-2019].

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

Nahas, Fatemeh. "A Brief Review of Neural Machine Translation (NMT)". In: *SAMPLE TRANSLATIONS* (), p. 15.

Robertson, Sean. *Translation with a Sequence to Sequence Network and Attention*. Ed. by PyTorch. `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`. [Online; accessed 17-April-2019]. 2017.

Rumelhart, David E et al. "Sequential thought processes in PDP models". In: *Parallel distributed processing: explorations in the microstructures of cognition* 2 (1986), pp. 3–57.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*. 2012.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

World, Ethnologue Languages of the, ed. *Summary by language size*. `https://www.ethnologue.com/statistics/size`. [Online; accessed 2-April-2019]. 2019.