Machine Translation

Ankit Wadhawan - Machine Learning Nanodegree, Capstone Project

27th September, 2018

I. DEFINITION

PROJECT OVERVIEW

Machine translation (MT) is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another. There have been multiple attempts to solve this problem via statistical/neural techniques in recent years, gaining success in varied degrees .

The approach I would be taking involves Deep Neural Machine Translation. Deep NMT is an extension of neural machine translation(NMT), which is an approach to machine translation that uses a large artificial neural network to predict the likelihood of a sequence of words . Both use a large neural network with the difference that deep neural machine translation processes multiple neural network layers instead of just one.

Translation from one language to another without human intervention has multiple critical applications across wide domains, and it is currently a very resource intensive field where machines have only just started reaching human levels on accuracy.

Industry leading translation systems provided by organisations such as Google have also started moving towards Neural Network based approaches (GNMT).

The aim of the project is to be able to translate English text, eg “I am John.”, to it’s French equivalent - in this case “Je suis John.”.

Typical productionised Language translation models are very resource intensive, with millions of sentences, words & large parallel corpuses required. This in turn requires expensive computing resources (the training needs to be done on large GPUs) & weeks of training.

To mitigate a few of these barriers, I will be using the dataset provided in the DLND language translation project by Udacity at Github, which includes a relatively small vocabulary & a limited amount of parallel sentences, hence not requiring a lot of computing power & time. It is like a minified parallel corpus and contains **137K** sentences, with each English sentence corresponding to a French sentence, per line in both files.

PROBLEM STATEMENT

The aim of the project is to translate sentences from a given language to another. The source language I would be using is English, with the target language being French.

I would be constructing a **word - based Sequence to Sequence Model** for achieving an NMT-based solution to the above problem statement. The tasks involved are the following:

1. Preprocess data by changing strings to vector representations (embeddings) & padding all sentences to be of the same length.
2. Create a Sequence to Sequence model using Tensorflow & train the model
3. Evaluate the results on the test dataset, and compute the BLEU score for finding out the accuracy of the model.

II. ANALYSIS

METRICS

The evaluation metric for this project would be the **BLEU Score**. BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. In other words, it is a metric for evaluating a generated sentence to a reference sentence.
A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

Suppose we have a translation ***T*** and its reference ***R***, BLEU is computed with precision *P(N, T, R)* and brevity penalty *BP(T,R)*.

$$\mathbf{BLEU = P \times BP}$$

where *P* is the geometric mean of n-gram precisions:

$$P(N,T,R) = \left(\prod_{n=1}^{N} p_n\right)^{\frac{1}{N}}$$

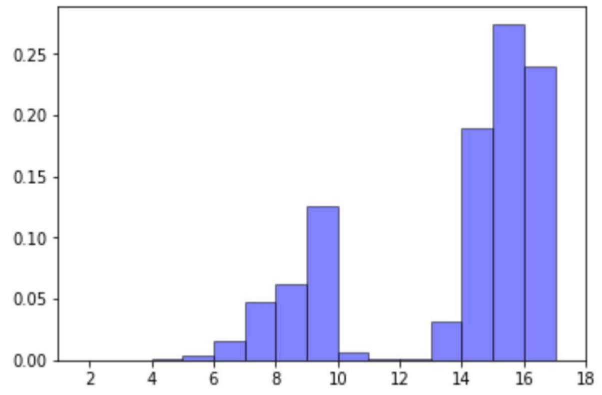
where: $p_n = m_n / l_n$. m_n is the number of matched n-grams between *T* and *R*, and l_n is the total number of n-grams in *T*.
BLEU's brevity penalty *(BP)* punishes the score if the translation length *len(T)* is shorter than the reference length *len(R)*, using this equation:

$$\text{BP}(T,R) = \min\left(1.0, \exp\left(1 - \frac{\text{len}(R)}{\text{len}(T)}\right)\right)$$

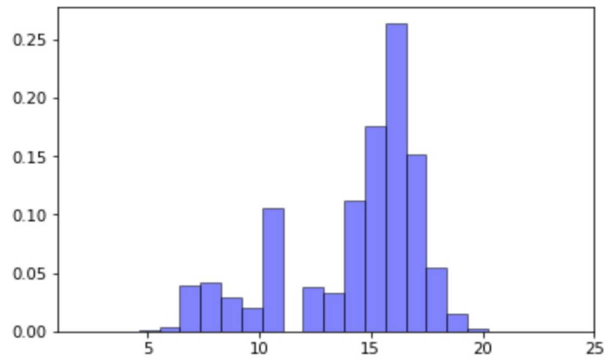
DATA EXPLORATION & VISUALISATION

The data set used in the project exists as a set of two files ‘small_vocab_en’ & ‘small_vocab_fr’. The dataset contains approximately **227** unique words in the source language (English) & **137K** sentences, with greater than **13** words on average in a sentence Plotting the distribution of all sentence lengths, and a few other stats:

```
Dataset Stats
small_vocab_en: English Sentences
Unique words: 227
Number of sentences: 137861
Average words / sentence: 13.225277634719028
```



```
Dataset Stats
small_vocab_fr: French Sentences
Unique words: 355
Number of sentences: 137861
Average words / sentence: 14.226612312401622
```



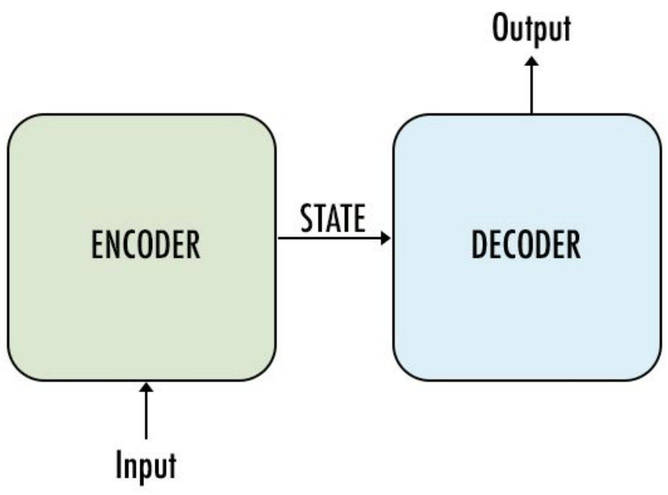
ALGORITHMS AND TECHNIQUES

LSTM

A Seq2Seq model is built on top of **Recurrent Neural Networks(RNNs).** An RNN is an artificial neural network where connections between nodes form a directed graph along a sequence. In simple terms, this means that the output of the RNN is fed back to it. This allows it to exhibit dynamic behavior for a time sequence. Unlike feedforward neural networks where no cycles are formed, RNNs can use their internal state (memory) to process sequences of inputs.

Since translation requires long term dependencies, we will be using a special type of RNN known as LSTM (Long Short Term Memory) Neural Network. LSTMs are a special kind of RNN which are capable of learning & storing long term dependencies.

Sequence to Sequence Model



A typical sequence to sequence model has two parts – an **encoder** and a **decoder**. Both the parts are practically two different recurrent neural network (RNN) models combined into one giant network.

Broadly, the task of an encoder network is to understand the input sequence, and create a smaller dimensional representation of it. This representation is then forwarded to a decoder network which generates a sequence of its own that represents the output.

Tensorflow tf.contrib APIs, among others will be used to build the Sequence to Sequence model. The following hyper-parameters will be used to build & tweak the network:

- **Epochs** - No. of times the model will be trained.
- **Batch size** - Batch size of the data (number of sentences) to be fed into the network. To be tweaked based on resources.
- **RNN Size -** No. of hidden layers in the RNN cell. Usually a greater size shows better performance.
- **Number of Layers** - No. of RNN layers in the encoder & decoder parts of the model.
- **Embedding size** - Word embedding sizes for the encoding & decoding layers.
- **Dropout layer keep probability** - Keep probability of the dropout layers in the encoder/decoder.
- **Learning rate** - learning rate of the network.

Embedding Layer

An embedding layer, is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as language modeling or document classification. It is a mapping from discrete objects (words) to vector representations.

The embedding layer is used to provide input to the encoder of a sequence to sequence network. To create word embeddings in Tensorflow, we split the source & target sentences into discrete words, and then assign an integer ID to every word in the vocabulary of the source/target respectively. Sentences are then fed to the encoder as their integer representations.

BENCHMARK

For the benchmark model, we will use the **character-level sequence-to-sequence model** provided by Keras, processing the input character-by-character and generating the output character-by-character.

https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py

III. METHODOLOGY

DATA PREPROCESSING

The dataset contains two files with **1,37,861** sentences, each line corresponding to its translation in the other file. The following operations are performed on the dataset to prepare the data for the embedding & the encoding layer:

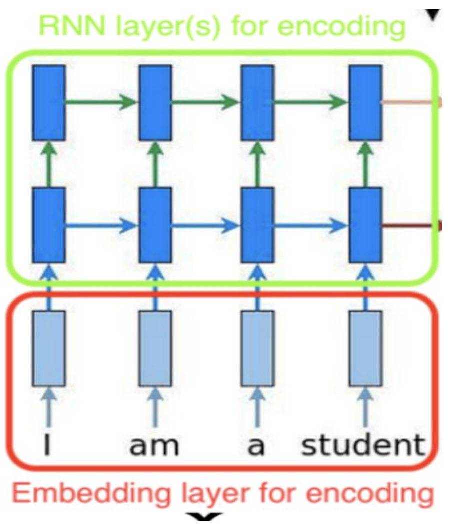
- Create vocabulary tables (look up dictionaries for word -> integer Ids & integer ids -> words).
- Preprocess data for the model: convert source & target sentences to a list of integer representations of the sentence's words. Add keyword <UNK> if unknown word is encountered.
- Batch & pad data: to keep the input batch dimensions consistent for the Neural Network (per batch), special keyword <PAD> is to be applied to all sentences in a batch to make their lengths equal to the largest sentence in the batch.
- Further processing: Use keywords such as <GO> to mark the start of every sentence, and <EOS> to mark the end of the sentence. This will aid the network in understanding the start & end of a sequence, respectively.

There are no outliers or anomalies in the data that need to be corrected, as seen in the data exploration section above.

IMPLEMENTATION

Since the sequence to sequence model is composed of two individual RNNs which act as encoder/decoder, we will be building two separate multi-layer RNN networks for the model, we the same hidden dimension size.

Encoding

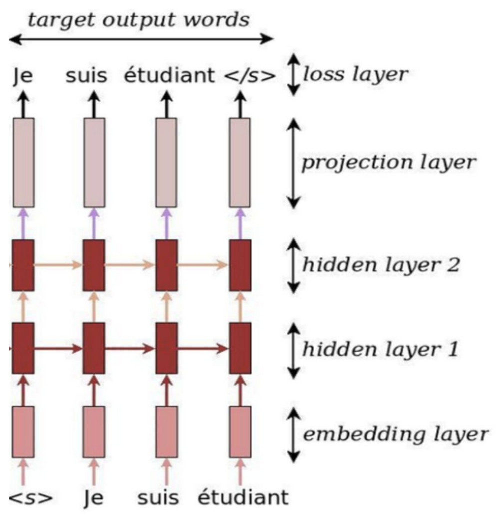


The encoding model consists of two different parts - the embedding layer & the RNN layer/layers.

- Embedding layer: Each word in a sentence will be represented with a word embedding that is computed by this layer. TF contrib.layers.embed_sequence API is used for this layer.
- RNN layer: We stack multiple LSTM cells together using tf.contrib.MultiRNNCell, after a Dropout Wrapper is applied with the keep probability parameter.

Decoding

The decoding component of the sequence to sequence model is made up of two separate processes, training and inference. They share the same architecture and its parameters, but have a different strategy to feed the shared model.



The training layer uses output from the encoder state & target sentences as its inputs and trains the model to predict the target sequences with given inputs.

The inference layer shares embeddings with the training layer. We use a manually created embedding parameter for the training phase, and convert the input to the embedding before the training is run.

For the inference process, whenever the output of the current time step is computed via the decoder, it will be embedded by the shared embedding parameter and become the input for the next time step.

We will also use a fully connected output layer that is used at the end of the entire process to map the outputs of the decoder to the elements of the target vocabulary, thereby ‘translating’ the sentence to the target language.

Sequence to Sequence

The combination of the above two encoding/decoding components, where the encoder state (that is returned after the encoding step) is used to feed the input to the decoding step to create the desired target sentence, constitutes the sequence to sequence model.

TF contrib.seq2seq.sequence_loss is used as the loss function, which computes the weighted softmax cross entropy loss, and is designed to be applied for RNNs. To optimize the model, I chose adam optimizer with the initial learning rate of 0.0002. Adam optimization works by maintaining a learning rate for each network parameter, which is separately adapted as learning unfolds. Adam optimizer is applied on the output of the seq2seq network to compute the gradient descent on the loss & train the model.

Benchmark

The Keras example for a character based sequence to sequence model, used as the benchmark model, was fed the same data as above, and the BLEU score was then computed on its outputs.

REFINEMENT

The initial model gave a low accuracy score. However, I was able to refine it with hit & trial. A few issues I faced were:

- Number of layers & RNN cell size: I initially set the rnn_size parameter as 512 & num_layers (number of RNN layers in the encoder/decoder networks) as 3, but the accuracy didn't improve a lot. This might be due to overfitting. Switching down to just 1 layer & rnn_size as 196 sped up the training while maintaining accuracy, which seems appropriate considering the small dataset being used.

- Really small batch size: Due to GPU resource constraints, I initially tried training on a small batch size of 20, with not so promising results. Since too large / too small batch sizes can have an adverse effect on training, I used colab's GPUs to train my models, ensuring a sufficiently apt batch size (256) & accuracy.

IV. RESULTS

MODEL EVALUATION & VALIDATION

The evaluation metric used for accuracy, as described above, was the BLEU score, computed using the corpus_bleu implementation from the Natural Language Toolkit library. The training accuracy **0.9527** after 3 epochs, whereas the validation accuracy was **0.9667.** The training set contains **110K** English - French sentence pairs.

Then, the model was run through a validation set of **27K** unknown sentences from the dataset (14.28% of the total data), that were split from the input before training. The validation score: **0.8962** .

The following hyperparameters were used to achieve the above score:

```
epochs = 3, batch_size = 256, rnn_size = 196,
num_layers = 1, encoding_embedding_size = 196,
decoding_embedding_size = 196, keep_probability = 0.75,
learning_rate = 0.002
```

1.0 indicates the perfect BLEU score. Since the data set was limited with a low vocabulary size, high training & validation accuracy was expected as a result. However, the model will not work too well outside the limited scope of the 227 unique words it learnt as its vocab embeddings. The only way to improve it is to train it with actual sized parallel corpus containing millions of sentences & a much larger vocabulary, such as the IIT-B English Hindi parallel Corpus .

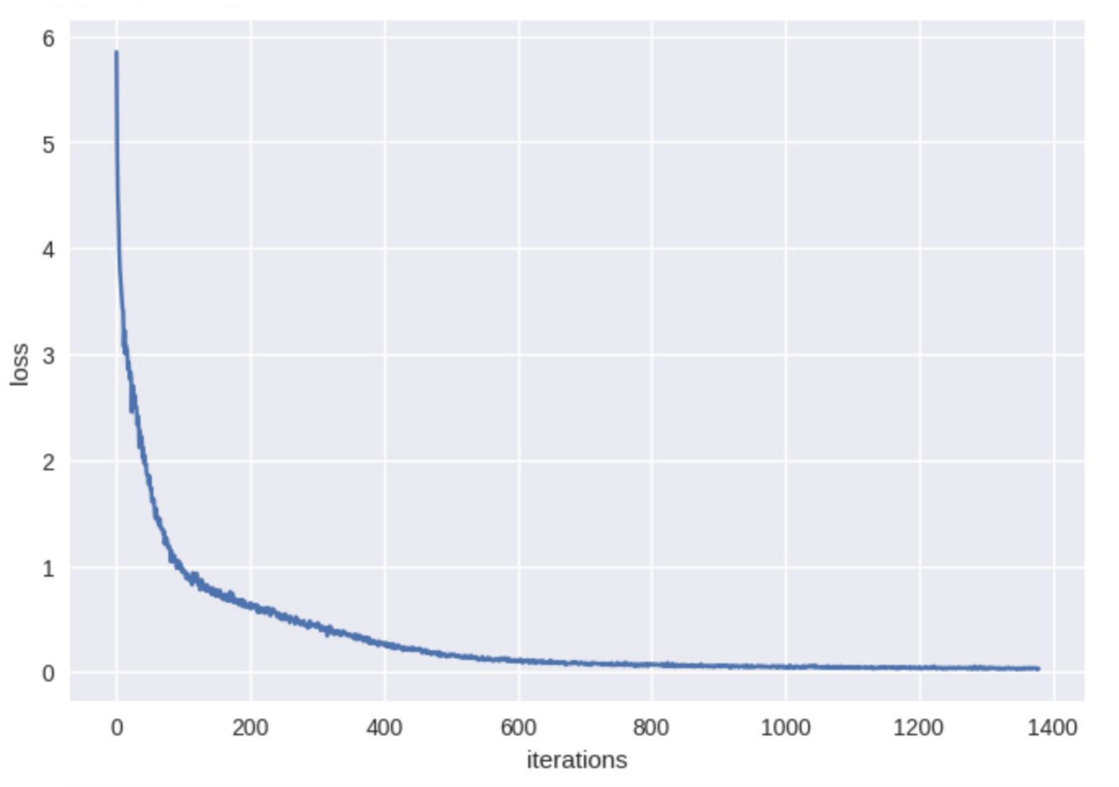
JUSTIFICATION

The Keras character based sequence to sequence model received an accuracy score of **0.1910** .

Since the average length of sentences was > 13 words, the character based model might not have been able to learn the larger context, as compared to the word embeddings based model.

V. CONCLUSION

FREE FORM VISUALIZATION



The above graph shows the loss value per training batch iteration, across 3 epochs. Initially the loss is a bit high, but as it learns the word embeddings the loss gradually goes down, gradually nearing 0.

REFLECTION

The intent of the project was to figure out whether a word based sequence to sequence model would do any better on a small embedding space, as compared to the Keras example that chose character based encoding-decoding. As I started with my initial implementation, a challenging part was to visualize the seq2seq model as a higher level model on top of RNN & LSTM, which were both very new to me.

As the project went along, I also got to learn about industry leading implementations in this domain, such as GNMT, and about tensorflow/keras & their respective APIs that aid working with language based models.

Due to a lack of dedicated, high performing GPUs, I was unable to dive very deep into training the model with a large corpus, which came with its own intricacies & decision making, and as a result, the model would mostly not perform well outside the scope of the 227 word vocabulary of the source language. However, it was interesting to read about & apply some of the recent advances in NMT, BLEU as a metric, and processing text data for input to the neural network, and creating my own embedding layer for the network input.

IMPROVEMENTS

There are quite a few improvements that can be done to improve the model and test it outside the bounds of the limited dataset used. Some of them are:

- Using Word2Vec or other existing embeddings instead of creating a local one, which would be having a highly limited set of data to build upon.
- Training the model on a large corpus of sentences, such as the WMT corpus or the IIT-B English Hindi Parallel corpus. Tweaking parameters & adding more layers to observe its effects on the model for training such a corpus would greatly improve the model
- Evaluate more advanced concepts such as Attention & Beam Search, using bi-directional RNNs to create a better seq2seq network.

VI. RESOURCES

https://arxiv.org/abs/1609.08144

https://arxiv.org/abs/1609.04836

https://en.wikipedia.org/wiki/Machine_translation

https://en.wikipedia.org/wiki/Recurrent_neural_network

https://github.com/udacity/deep-learning/tree/master/language-translation/data

https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py

https://en.wikipedia.org/wiki/BLEU

http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdf

https://machinelearningmastery.com/diagnose-overfitting-underfitting-lstm-models/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html