Translation with a Sequence to Sequence Network

Melissa Lynch

July 31, 2020

1 Introduction

In this paper, I present two models trained to translate from English to Spanish and Spanish to English. This is made possible by the powerful idea of a sequence to sequence network, in which two recurrent neural networks (RNNs) work together to transform one sequence to another. This neural network design pattern is also called encoder-decoder architecture. As the name suggests, the network architecture is partitioned into two parts, the encoder and the decoder. The encoder's role is to encode the inputs into state. Then the state is passed into the decoder to generate the outputs. The encoder is a normal neural network that takes inputs (such as a source sentence) to return outputs. An encoder network condenses an input sequence into a vector, and a decoder network unfolds that vector into a new sequence.

In the context of machine translation, the encoder transforms a source sentence into state (represented as a vector) that captures its semantic information. The decoder then uses this state to generate the translated target sentence. To improve upon the basic decoder model, an attention mechanism is also used; this lets the decoder learn to focus over a specific range of the input sequence.

2 Datasets

This model can be used with any language pairs, assuming it's given the appropriate training data. For this project, I chose to train the model with two language pairs: English to Spanish and Spanish to English. Even though I was able to use the same data for training both models, they are completely independent of each other. The motivation for choosing this language pair was 1) English and Spanish are the languages in which I am most proficient, and 2) English and Spanish are not too different, have similar characters, and there is a lot of data for these languages. I ended up using parallel corpora from OpenSubtitles (http://www.opensubtitles.org/) to train the models. OpenSubtitles is a collection of translated movie subtitles with many language pairs. Their English-Spanish parallel corpus consisted of over 40 million sentence pairs; I trimmed this down to 250,000 pairs.

The first data set I tried was Europarl Parallel Corpus. Unfortunately, the results were not great. The language being used was too formal and not like the short, conversational translations I intended the model to work for. I did not need to do any official accuracy tests to know it wasn't going to work out. This was a little surprising to me, as Europarl is known to be a reliable parallel corpus for statistical machine translation. The OpenSubtitles corpus was a much better fit for the task.

Loading Data

To encode the training data, each word in a language is represented as as a one-hot vector, or giant vector of zeros except for a single one (at the index of the word). These encoding vectors can be quite large, especially when working with larger data sets. For example, in 167,748 sentence pairs there were 32,929 Spanish words and 23,349 English words. The number of words in a language is the size of the encoding vector. A unique index per word is needed to use as the inputs and targets of the networks later. To keep track of all this, a helper class called Lang is used to represent each language. The class will store word $\rightarrow$ index (word2index) and index $\rightarrow$ word (index2word) dictionaries.

Since the training data is raw text, the first processing step is to ensure any Unicode characters are converted to ASCII. Then, all characters are made lowercase and most punctuation is trimmed. The input sentence length is limited to 10 words. All English sentence examples are in one file and Spanish examples are in another. The examples are matched by line number. For instance, the English example on line 23 in the English file has the appropriate Spanish translation on line 23 of the Spanish file. Each language file is read line by line. The English lines and Spanish lines are then split into pairs. The pairs are directional, that is, the first sentence in the pair is the in source language and the second sentence is in the target language. Whether the pairs will be English to Spanish or Spanish to English is handled by simply using a command line argument when running the program. After the sentence pairs are processed and filtered, a Lang object is created for each language using these pairs.

3 Sequence to Sequence Model

A sequence to sequence network is a neural network design pattern in which two recurrent neural networks (RNNs) work together to transform one sequence to another. An RNN is a network that operates on a sequence and uses its own output as input for subsequent steps. One RNN is called the encoder, and the other is called the decoder. The encoder reads an input sequence and outputs a single vector, and the decoder reads that vector to produce an output sequence. Unlike sequence prediction with a single RNN, where every input corresponds to an output, the encoder-decoder model frees us from sequence length and order, which makes it ideal for translation between two languages.

Consider the sentence "No soy el gato negro" $\rightarrow$ "I am not the black cat".

Most of the words in the input sentence have a direct translation in the output sentence, but are in slightly different orders, e.g. "gato negro" and "black cat". Because Spanish is a null-subject language, the subject "I" is not present in the input sentence, meaning there is one more word in the output sentence. These factors would make it difficult to produce a correct translation directly from the sequence of input words.

The Encoder

The encoder of a sequence to sequence network is an RNN that outputs some value for every word from the input sentence. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word. The job of the encoder is to, ideally, encode the "meaning" of the input sequence into a single vector — a single point in some N dimensional space of sentences.

The Decoder

The decoder is another RNN that takes the encoder output vector and outputs a sequence of words to create the translation. The simplest decoder would only use the last output of the encoder. This last output is sometimes called the context vector as it encodes context from the entire sequence. This context vector is used as the initial hidden state of the decoder. At every step of decoding, the decoder is given an input token and hidden state. The initial input token is the start-of-string (SOS) token, and the first hidden state is the context vector (the encoder's last hidden state). If only the context vector is passed between the encoder and decoder, that single vector carries the burden of encoding the entire sentence. To improve the accuracy of the model, a decoder with attention was used.

Attention allows the decoder network to focus on a different part of the encoder's outputs for every step of the decoder's own outputs. First, a set of attention weights is calculated. Calculating the attention weights is done with another feed-forward layer, using the decoder's input and hidden state as inputs. These weights are then multiplied by the encoder output vectors to create a weighted combination. The result (called attn_applied in the code) should contain information about that specific part of the input sequence, and thus help the decoder choose the right output words.

4 Training and Evaluation

Training the model was done via Google Cloud Platform. The model took around 15 hours to train using one NVIDIA Tesla K80 GPU. Because English to Spanish and Spanish to English are separate models, total training time was around 30 hours. The training data contained 250,000 sentence pairs; after

filtering the data, 167,748 samples were trained over 5 epochs. An epoch is one complete pass through the training dataset.

Preparing Training Data

Before training a sample, an input tensor (indices of the words in the input sentence) and target tensor (indices of the words in the target sentence) need to be created. Using the word2index dictionaries, the raw sentence data is transformed into a vector of unique indices representing each word. The EOS token is appended to the end of this vector. A tensor is then created from this vector of indices. This is done for both the input and the target sentence in a given example.

Training the Model

To train a single example, the input sentence is first run through the encoder. Each output from the encoder and the latest hidden state are stored. Then the decoder is given the SOS token as its first input, and the last hidden state of the encoder as its first hidden state. The decoder always takes in an input word, the hidden layer, and the encoder outputs. Where the decoder gets its input from can vary. "Teacher forcing" is the concept of using the real target outputs as each next input, instead of using the decoder's guess as the next input. Using teacher forcing causes the model to converge faster. The model chooses randomly whether or not to use teacher forcing; I kept the ratio at 0.5, so teacher forcing was used around 50 percent of the time during training.

For the model to successfully learn, train is called many times in another function called trainEpoch. This is also where the optimizers for the encoder and decoder are initialized, as well as the loss function. The optimizer is stochastic gradient descent and the loss function is negative log likelihood loss. The model was trained over 5 epochs, meaning the entire dataset was passed through the network 5 times.

Evaluation

Evaluation is mostly the same as training, but there are no targets so the decoder's predictions are fed back to itself for each step. Each time a word is predicted it is added to the output string, and if the EOS token is predicted then the evaluation stops.

5 Accuracy and Results

The goal of this project was to make a versatile translation system. The very first iteration of this project was trained on a very small data set of about 7,000 examples and took around 45 minutes to train on a MacBook CPU. It was only able to handle simple inputs such as: I am happy, she is pretty, they are sad,

etc. The final version of the model can handle not only inputs of that format, but also a wide variety of sentences.

BLEU Score

To test the accuracy of the translations being produced, I took the average BLEU score of 25,000 examples. BLEU is a score for comparing a candidate translation of text to one or more reference translations. For simplicity, I only used one reference and that was the target sentence for a given example; the candidate is the model's prediction. In one test I used the training data and in the other I used new/unseen data. BLEU scores were calculated using NLTK (Natural Language Toolkit). NLTK provides the sentence_bleu() function for evaluating a candidate sentence against one or more reference sentences. Because the word2index and index2word dictionaries are built using the training data, some words in new examples are unknown. In these cases, the whole example is skipped and not considered in the averages. Since the training data is diverse this doesn't happen too often; for 25,000 attempts, 21,305 of those were successful. Below are the results:

English to Spanish

	1-gram	2-gram	3-gram	4-gram
Training data	0.4589	0.2471	0.1070	0.0392
New data	0.4082	0.1850	0.0632	0.0165

Spanish to English

	1-gram	2-gram	3-gram	4-gram
Training data	0.4683	0.2883	0.1556	0.07834
New data	0.4176	0.2315	0.1161	0.0517

Human Judgement

While using BLEU scores is a convenient way to automatically test the accuracy of translation systems, it has many shortcomings as well. There are many examples where a translation has a low score, and it is actually correct. In all of the following examples, the predictions are fully accurate but do not match the target sentence exactly:

Input: what do you say ?
Target: que dice ?
Prediction: que dices?

Input: how re you ?
Target: como te va ?
Prediction: como estas ?

Input: one more horse !
Target: otro caballo mas !
Prediction: un mas caballo !

Input: the men are in a room .
Target: los dos estan en una habitacion .
Prediction: los hombres estan en habitacion .

Another drawback, although less of an issue, is that for examples that only have two words, the 3-gram and 4-gram scores will be 0; for a sample that has only three words, the 4-gram score will be 0. Many examples from the data can be quite short, only two or three words, so this can negatively impact higher n-gram scores.

Furthermore, the training data is not perfect. Sometimes the targets are not actually proper translations. Take the following for example:

Input: yes you did !
Target: como sabes que el no lo hizo ?
Prediction: si lo hiciste !

The target translates to "How do you know he didn't?" whereas the prediction is the direct translation of the input.

There are, of course, genuinely bad translations as well:

Input: it's payday for dad .
Target: porque hoy le pagan a papa .
Prediction: es la papa de papa .

Input: their famous chocolate shake ?
Target: el batido de chocolate de siempre ?
Prediction: el chocolate de chocolate de chocolate ?

I ran an experiment calculating BLEU score averages of scores above 0.05 only. I'm not sure exactly what this represents (if anything) but I thought it would be interesting to share the data nonetheless. I ran 25,000 examples of training data and new data. For the training data, the actual number of examples that met that threshold is as follows (from 1-gram to 4-gram): [20638, 11751, 5579, 2486]. For the new data it is slightly less, since some examples are skipped due to unknown input words: 20468 total, [16049, 8170, 3608, 1394].

English to Spanish

	1-gram	2-gram	3-gram	4-gram
Training data	0.5852	0.6484	0.7434	0.8365
New data	0.5442	0.6080	0.7145	0.8022

Spanish to English

	1-gram	2-gram	3-gram	4-gram
Training data	0.5727	0.6094	0.6916	0.7745
New data	0.5360	0.5781	0.6866	0.7811

6 Conclusion

Using the idea of a sequence to sequence network, I was able to train a system that can translate from one language to another with relative success. The baseline model was very limited in that it could only handle inputs of a certain form. By expanding the dataset and the training time, I was able to transform the model into an advanced machine translation system that can handle a variety of inputs. Although the models presented in this paper were from translating from English to Spanish and Spanish to English, the same approach can be used with any language pairs.