Published in Towards Data Science

Eileen Pangu Follow
Jan 22 · 6 min read · · Listen

End-to-End Attention-Based Machine Translation Model with Minimum Tensorflow Code

This blog post walks through the training and inference of attention-based machine translation using only high level Tensorflow API. It's a simplified end to end tutorial that aims to minimize lines of code.

There are a lot of online tutorials on this topic. But most of them they're all variants of the Tensorflow official tutorial, which includes a huge amount of boilerplate code. While it's helpful to understand how it's implemented using low level API, a readily available and simplified implementation is often appreciated — beginners can use it to quickly form an intuition of the subject; practitioners can easily borrow the self-contained code snippets. This blog post offers a simplified end to end implementation of attention-based machine translation code, implemented using only high level Tensorflow Keras layers.

> *Translation Task*

The translation task we'll perform i 104 n the Tensorflow official tutorial.

```
$ curl http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip -output spa-eng.zip
$ unzip spa-eng.zip
$ pip install -user tensorflow-text
```

Preprocessing

Text preprocessing is an integral step in any natural language post. If you're interested in seeing the impact of various text preprocessing techniques, feel free to check out this blog post for more details. We leverage the `TextLineDataset` API to directly construct the datasets, avoiding much of the boilerplate code in the Tensorflow official tutorial.

```python
import tensorflow as tf

def split(text):
    parts = tf.strings.split(text, sep='\t')
    return parts[0], parts[1]

 dataset = tf.data.TextLineDataset(['spa.txt']).map(split)
eng_dataset = dataset.map(lambda eng, spa : eng)
spa_dataset = dataset.map(lambda eng, spa : spa)
```

translation_load.py hosted with ❤ by GitHub view raw

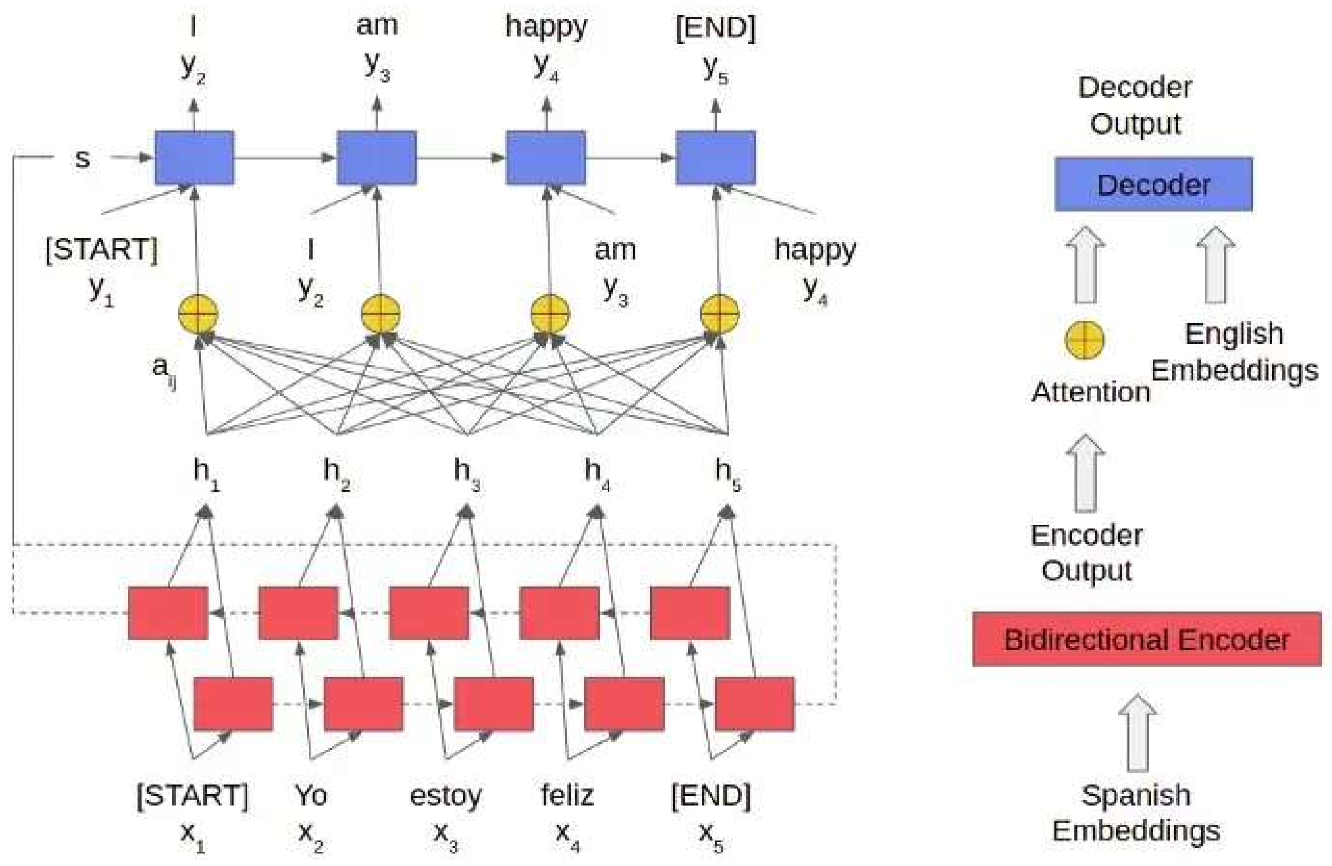
Data Loading

Now the datasets are ready to use, we employ the `TextVectorization` layer to index the tokens in the text. This part is almost identical to the Tensorflow official tutorial. The goal is to construct the respective vocabulary sets for Spanish and English, so that we can map tokens to indices, which will be used later on for embedding.

Text Preprocessing

Model Architecture

Here comes the meat of this blog post: the model. Before going into the model code, let's have a look at the high level view of the model architecture. The model is a typical sequence to sequence model. The Spanish input is fed into a bidirectional recurrent neural network, which we call the bidirectional encoder. The final state of the bidirectional encoder `s` becomes the initial state of a decoder. The output of the bidirectional encoder `h` is weighed by an attention layer `aij` and combined with the English input. We use the teacher forcing mechanism to train the decoder, i.e, the English input to the decoder comes from the expected (instead of actual) decoder easier. The decode outputs the final English. See the following figure for an overview.



Model Architecture

Note that the English input to the decoder has the `[START]` token and the English output of the decode has the `[END]` token. One is used to kick off the text generation, the other is used to terminate the text generation. The English input/output content is also shift one step. The Spanish input to the bidirectional encoder has both the `[START]` and `[END]` tokens. That's simply because we reuse the same `standardize` function for both languages. In theory, the Spanish input does not need those two tokens.

During inference time, we no longer have the expected English as input. So we'll have to use the decoder's output from the previous step, and generate the output sequence one step at a time.

Model Interface

Inside the initialization, we construct the necessary layers for the model, including the `TextVectorization` layers for both Spanish and English, the `Embedding` layers for both Spanish and English, the `Bidirectional` layer for Spanish encoder, the `LSTM` layer for

Model Initialization

A couple noteworthy points: the `Bidirectional` layer's internal `LSTM` has half of the unit as other layers because the `Bidirectional` layer will concatenate the forward and backward outputs. Both the `Bidirectional` encoder and `LSTM` decoder are set to return

The model invocation calls the layers and returns the output. See the following code snippet for the call function.

Model Invocation

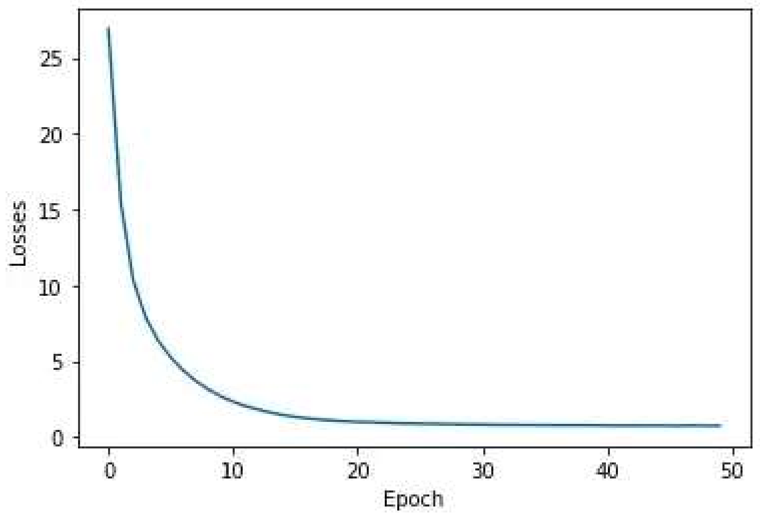
of the decoder is `(batch, Te-1, rnn_output_dim)`. The final output `Dense` layer of `eng_vocab_size` units is smart enough to produce output of shape `(batch, Te-1, eng_vocab_size)`. The masks, originally created by the embedding layers, are passed through all layers, sometimes implicitly (in the recurrent neural networks) and other times explicitly (in the Attention layer and final function return).

Training Code

The training code iterates through batches of sentence pairs, collects the output from the model, calculates the loss, and applies gradient descent. See the following code snippet for the training part. The most significant bit is that in the loss calculation, we need to mask out the unrelated elements — because they're just paddings for the variable length sentences.

Model Training

After 50 epochs, we can see the model loss is approaching zero. So that means our model has the capacity to overfit the data, which is a good indicator of the model power. A lot of regularization techniques could be applied here to safeguard the model's generalization capability. But it's out of scope for this blog post.



Training Loss

> ***Inference Code***

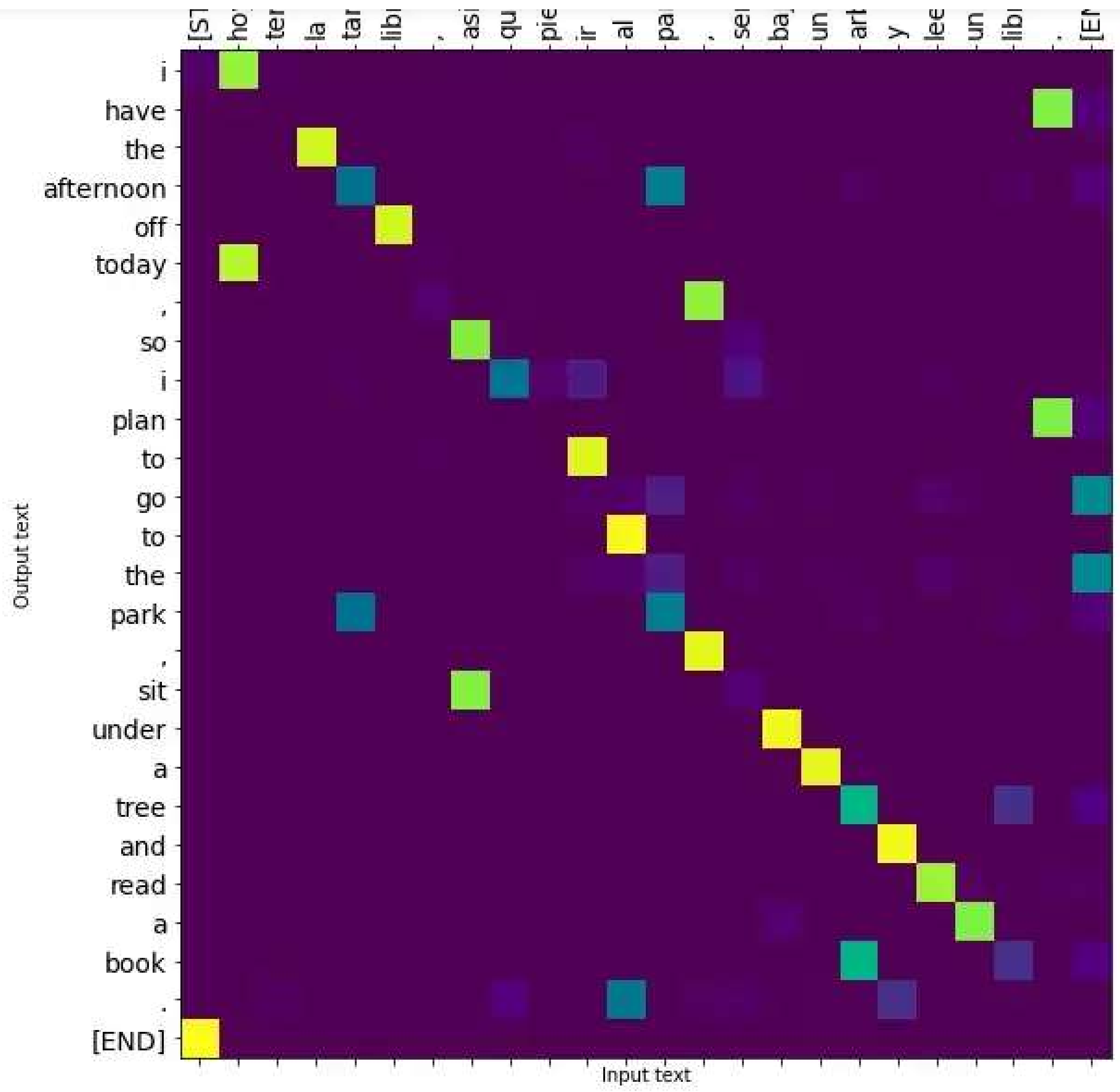
The inference code is very similar to the model invocation code. The key difference is that instead of generating the English output all at once, we need to generate it one step at a time during inference because we need the decoder output from the previous step. See the following code snippet for the inference code. Note that the inference code also returns the attention matrix, which we can use to visualize the model attention between the Spanish input and English output.

Model Inference

Last but not least, let’s call the `translate` inference function for the randomly picked Spanish input `“Hoy tengo la tarde libre, así que pienso ir al parque, sentarme bajo un árbol y leer un libro.”`. The output is `“i have the afternoon off today , so i plan to go to the park , sit under a tree and read a book .”`, which is exactly the same as the English output in the dataset, minus the letter case and punctuation space due to our text preprocessing. No surprise. As we discussed above, the model is overfitting. Let’s also visualize the attention using a `plot_attention` function similar to the helper function in the Tensorflow official tutorial.

Attention Plotting

Open in app Get started



Attention Plot

Thanks to Ludovic Benistant