Technology Review: NLP with Tensorflow

Xue Ying Lin (xylin2@)

Department of Computer Science, University of Illinois at Urbana-Champaign

CS 410: Text Information Systems

Dr. Cheng Xiang Zhai

November 7, 2021

Technology Review: NLP with Tensorflow

The state of the art in Natural Language Processing (NLP) has significantly advanced in the last several years from an active area of research to a set of practical tools and techniques used for a variety of complex applications. Many of these applications can elevate the interface between computers and humans to a new level of understanding and capability.

The use of Recurrent Neural Networks (RNNs) and machine learning specifically have emerged in the last few years as key approaches to improving NLP, particularly in areas like sentiment analysis, and entity recognition. TensorFlow is a popular open-source framework consisting of tools and libraries to power ML applications, and helps in many applications of NLP.

This review provides an overview of some of the components Tensorflow provides to aid in sentiment analysis and named entity encoding as described in (Bansal, 2021), and how these can be used alongside tools like Scikit-Learn and Keras (Géron, 2019).

Natural Language Understanding and RNNs

Natural Language Understanding (NLU) is a subfield of NLP, and an area of significant interest because of the many practical applications it has, and because of the complexity of the problem space.

Imagine a system where we attempt to categorize messages in a student discussion forum by topic. We can achieve a first approximation of this by looking at common repeating words. However, doing this at scale and accurately requires understanding the nuances of how parts of speech refer to entities and each other. (Bansal, 2021) gives an example sentence: *"A CNN helps improve the accuracy of object recognition"*, and notes that without effective NLU, we might infer that this sentence is talking about a news network rather than machine learning.

A common problem is providing sufficient NLU that we can identify the positive or negative sentiment of a block of previously unseen text.

Recurrent neural networks (RNNs) emerged in the 1980s and are able to be dynamically trained on deep learning problems. However, a significant advance came in 1997 with the invention of Long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997). RNNs are fundamentally concerned with sequences or lists, and LSTMs introduce a cell state which can be used to effectively support long term dependency collection (Olah, 2015).

Despite the improvements, building and applying an RNN to NLU problems remains challenging and complex (Bansal, 2021), something frameworks like Tensorflow aim to simplify significantly.

Using Tensorflow for NLU

Almost all NLU cases involve building a model using *features* and *classes*. Statistical information about documents in our dataset such as the number of symbols in a given document are examples of features. A *sentiment* expressed as 0 for negative and 1 for positive is an example of a class. We usually also use a test dataset to evaluate our model for correctness.

Tensorflow provides the `tensorflow_datasets` package to allow us to easily load common datasets. Unlike simpler tools like `pandas`, we can load these models efficiently even if they're quite large, they don't necessarily have to be loaded entirely in-memory (TensorFlow Datasets, 2021). Here's how we'd load the `sentiment140` dataset, which consists of Tweets from the Twitter social network with associated data:

```
import tensorflow as tf
import tensorflow_datasets as tfds

twitter_train = tfds.load(name='sentiment140',
                          split='train', as_supervised=True)
twitter_test = tfds.load(name='sentiment140',
                         split='test', as_supervised=True)
```

The loaded dataset is a tuple of *tensors* which hold data. We can see the `text` and `polarity` of the first loaded data item (polarity is 0 =negative, 2 = neutral, 4 = positive):

```
for text, polarity in twitter_train.take(1):
  print(f"{text}\n{polarity}")

b"i'm 10x cooler than all of you! "
4
```

To actually build a model with this data, we need to normalize and tokenize the text. TensorFlow provides first class support for text processing in the TensorFlow Text library (TensorFlow Text, 2021). For example, to tokenize and build a vocabulary set:

```
import tensorflow_text as tf_text

tokenizer = tf_text.WhitespaceTokenizer()
vocab = set()
for text, polarity in twitter_train:
   tokens = tokenizer.tokenize(text.numpy()).numpy()
   vocab.update(tokens)
```

To actually efficiently extract features from the text, because the data set may be large, we supply functions to TensorFlow. We can specify that it runs these functions in parallel to provide more throughput while processing.

```
from tensorflow.keras.preprocessing import sequence

def pad_transform(sample):
   encoded = twitter_encoder.encode(sample.numpy())
   pad = sequence.pad_sequences([encoded], padding='post',
                                maxlen=150)
   return np.array(pad[0], dtype=np.int64)


def encode_fn(sample, label):
   encoded = tf.py_function(pad_transform,
                            inp=[sample],
                            Tout=(tf.int64))
   encoded.set_shape([None])
   label.set_shape([])
   return encoded, label
```

Given this function, we can encode the whole data set using map. The AUTOTUNE parameter turns on experimental automatic subprocess execution to speed up the processing:

```
encoded_train = twitter_train.map(encode_fn,
      num_parallel_calls=tf.data.experimental.AUTOTUNE)
encoded_test = twitter_test.map(encode_fn,
      num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

Finally, we can build an LSTM model using the tf.keras package and compile and fit the model:

```
model = tf.keras.Sequential([
   tf.keras.layers.Embedding(vocab_size, embedding_dim,
                             mask_zero=True,
                             batch_input_shape=[batch_size, None]),
   tf.keras.layers.LSTM(rnn_units),
   tf.keras.layers.Dense(1, activation='sigmoid')
 ])
model.compile(loss='binary_crossentropy',
            optimizer='adam',
            metrics=['accuracy', 'Precision', 'Recall'])
encoded_train_batched = encoded_train.batch(BATCH_SIZE)
model.fit(encoded_train_batched, epochs=10)

# Compare to the test set
model.evaluate(encoded_test.batch(BATCH_SIZE))
```

There are several tunable parameters in this model, including the size of the vocabulary, the number of units in the RNN, and the size of batches to process. When we compare the end result to the test set, we see about a 99% accuracy of recall on the original dataset, but only 92% on the test set, suggesting some overfitting. There's more work we could do to refine this model, but it's a good start.

Summary

I've only scratched the surface in this review, but we've seen that TensorFlow provides an end to end set of tools we can use to perform both text transformation on large data sets, and build RNNs with LSTM to perform sentiment analysis. With some adjustment, a similar model can

be used for other categories of NLU problem, and TensorFlow provides a wide number of capabilities that work hand in hand with other libraries such as Scikit-Learn and Keras to help build applications for many core NLP problems.

References

Bansal, A. (2021). *Advanced Natural Language Processing with Tensorflow 2*. Packt Publishing Ltd.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media Inc.

Hochreiter, Sepp; Schmidhuber, Jurgen. Long Short-Term Memory. *Neural Computation*, 9 (8): 1735-1780. http://dx.doi.org/10.1162%2Fneco.1997.9.8.1735

Olah, Christopher. (2015, August 27). *Understanding LSM Networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TensorFlow Datasets. (2021, Oct 22). https://www.tensorflow.org/datasets/catalog/overview

TensorFlow Text (2021). https://github.com/tensorflow/text