CS 410

Technology Review

Casey Blair

Fall 2020

Tensorflow VS PyTorch for Text Sentiment Analysis: A Comparison

This technology review will compare using Tensorflow and PyTorch for sentiment analysis: specifically, using neural networks and sub-word tokenization along with RNN and LTSM neural network layers to increase the accuracy of the sentiment prediction.

Tensorflow was created by Google and released in 2015 under the Apache Creative Commons license (Martín Abadi et al.). Tensorflow focuses on deep neural networks and, in its API, implements CNN (convolutional Neural Network), RNN (Recurrent Neural Network), and LTSM (Long Short Term Memory). Tensorflow allows the user to declare multiple layers in the neural network in a list: the developer can mix RNN, LTSM, and CNN layers to create a neural network for their use case.

PyTorch was developed by Facebook and published in 2018 with a BSD license (Peng, T.). PyTorch also includes CNN, RNN, and LTSM neural network layers in its API. Like Tensorflow, PyTorch allows the user to mix and match different neural network layer types in a single model. Both API's can use the GPU faster execution computation.

Both PyTorch and Tensorflow are Python libraries and are optimized for neural network creation as they focus on tensors, or multi-dimensional arrays. Both libraries provide an API for creating multi-layer neural networks. However, PyTorch's API syntax is slightly less verbose than Tensorflow syntactically: it takes less lines of code in PyTorch to declare the same multi-layer LTSM for sentiment analysis. Both libraries can use the GPU of the host machine for faster execution, and both are Python libraries that are compatible with the latest version of Python so integrating them with production code is easy and secure.

Creating training and test datasets for sentiment analysis can be a daunting and time-consuming task: especially for a single person such as a developer who just created their first neural network for sentiment analysis and needs data to train their model with. Luckily, both Tensorflow and PyTorch have libraries with pre-compiled datasets for training and testing purposes to reduce the burden on developers to create datasets.

Tensorflow's tesorflow_datasets library has pre-compiled datasets that can be used with Tensorflow TensorFlow Datasets 2020). The text datasets have pre-tokenized training and validation data that allow learners to focus on the neural network code while they are first learning the API or just want a common dataset to benchmark performance across different neural network implementations. Also, some of the datasets like the IMDB Movie Review dataset has a large labeled dataset with binary sentiment labels with 50,000 entries (Imdb_reviews). There are 50,000 entries in the dataset, and they are pre-split in to test and train sections for supervised learning purposes. Also, for unsupervised learning purposes, there is an unsupervised version of the data which groups all the data in to one set (Imdb_reviews).

Tensorflow's IMDB Movie Review dataset does come with a couple different versions of word tokenization. The default tokenization of the text data uses entire words as a token. However, there is also a sub-word tokenized version: instead of entire words being the token, the word is split into sections and those sections are the tokens. Sub-word tokenization, when used with sentiment analysis, can inprove the sentiment prediction since some words have the same root word so they have a similar meaning, but are classified as different tokens when the entire word is used as the token. For example, the word "run" and "running" have almost the same meaning: the difference is one is happening in the present, and one happens in the future. However, the action performed is the same. If we were able to split the word "running" in to "run" and "ning" then the sub-word token "run" would be the same as the first word's token "run" and the model would recognize the similar meaning.

PyTorch also has a similar dataset from IMDB movie reviews available in the torchnlp library (Source code for torchnlp.datasets.imdb.). PyTorch's IMDB dataset is also pre-split in to test and train sections. However, PyTorch's IMDB review dataset does not come with a sub-word tokenized version of the dataset. This means the developer would need to implement their own sub-word tokenizer and re-tokenize the dataset to have a sub-word dataset comparable to the one already provided in the Tensorflow dataset.

These large pre-compiled datasets available for sentiment analysis will save the developer a massive amount of time because creating a dataset for sentiment analysis requires manually tagging sentences with labels. Also, the work necessary to compile enough data to train a meaningful model is a substantial time commitment which is difficult for someone just learning about neural networks to undertake. Having these common use datasets readily available for developers that are learning Tensorflow's API or PyTorch's API to focus on their code instead of creating the dataset.

When developers wish to create their own text datasets, both Tensorflow and PyTorch have API elements for sub-word tokenization. Tensorflow's subword tokenization API is called a WordPiece tokenizer (Neale, R.). There is a subword tokenizer for PyTorch too, however, it's packaged in a separate library called "pytorch-nlp" (PetrochukM.).

For predicting sentiment on sequences of words, the developer needs to use neural network layers that can remember state from one sequence element to another since position and order matter. Recurrent Neural Networks are great for sentiment analysis as they can "remember" sequences. They pass the output of the previous node to the next node as an input. So the output of the previous word's node is used as input for the next node along with the next word in the sentence. Since the ordering of the input vector now affects subsequent nodes in the layer, this allows the vector's order to become a parameter in the network (Amidi, B., & Amidi, S.).

For sentiment analysis, predicting if a sentence is positive or negative, it's not enough to tokenize the sentence and check for the presence of words deemed positive or negative: syntactic analysis of the sentence's meaning is needed to determine if the sentences meaning is positive or negative. The ordering of the same words can affect the overall meaning of a sentence. Because of this, Recurrent Neural networks work well for sentiment analysis. However, RNN's have a potential problem when backpropagating weight updates which cause the calculated error to either explode exponentially or vanish completely (becomes zero) (Hochreiter, Sepp & Schmidhuber, Jürgen). Because of this, more effort was put in to RNN development to solve this issue and Long Short-Term Memory neural network layers were discovered (Hochreiter, Sepp & Schmidhuber, Jürgen).

LTSM network layers can pass node output from any word's node as input in to the current node: not just the word immediately preceeding the current word (Hochreiter, Sepp & Schmidhuber, Jürgen). This allows for more complex natural language syntax to be represented in the neural network layers as natural language sentences commonly have words that are not immediately next to each other in word order affect the meaning of each other. Because of the complexity of natural language it is useful to use more than just the previous word in the sequence to predict the current word's meaning. Because of this need, LTSM's were invented and are used in sentiment analysis.

Both PyTorch and Tensorflow have LTSM layers in their API. Integrating LTSM layers in with RNN, CNN, and other layer types is quite simple: in Tensorflow and PyTorch you declare the layer type in the list of layers for your neural network. LTSM layers do take more time to train then RNN's, so keep that in mind, and will give moderate to large prediction accuracy gains depending on the complexity of the text data's language syntax. If the text data is simple, then LTSM's minor accuracy gain may not be worth the extra computation time to train a more complex model, so keep that in mind when choosing which layer to use in your network.

PyTorch and Tensorflow both give powerful neural network capabilities to developers in only a couple lines. While Tensorflow has been around longer and more features in it's API, PyTorch's simple syntax make the barrier to learning much lower. Both API's have great tutorials on their respective home pages and both come with pre-compiled datasets to allow new learners to focus on code and not dataset creation.

Citations

Amidi, B., & Amidi, S. (n.d.). Recurrent Neural Networks cheatsheet Star. Retrieved November 15, 2020, from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

Imdb_reviews： TensorFlow Datasets. (2020, November 07). Retrieved November 15, 2020, from https://www.tensorflow.org/datasets/catalog/imdb_reviews

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Leven berg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens,Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. tensorflow.org.

Neale, R. (2019, June 10). Introducing TF.Text. Retrieved November 15, 2020, from https://blog.tensorflow.org/2019/06/introducing-tftext.html

Peng, T. (2018, May 16). Caffe2 Merges With PyTorch. Retrieved November 15, 2020, from https://medium.com/@Synced/caffe2-merges-with-pytorch-a89c70ad9eb7

PetrochukM. (n.d.). PetrochukM/PyTorch-NLP. Retrieved November 15, 2020, from https://github.com/PetrochukM/PyTorch-NLP

Source code for torchnlp.datasets.imdb. (n.d.). Retrieved November 15, 2020, from https://pytorchnlp.readthedocs.io/en/latest/_modules/torchnlp/datasets/imdb.html

TensorFlow Datasets. (2020, November 11). Retrieved November 15, 2020, from https://www.tensorflow.org/datasets/catalog/overview

Tf.keras.preprocessing.text.Tokenizer： TensorFlow Core v2.3.0. (n.d.). Retrieved November 15, 2020, from https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer