Some Applications of Deep Learning to Problems in Natural Language Processing.

May 18, 2024

Contents

1 Introduction

This repo contains implementations of deep learning techniques to problems in natural language processing. Two different datasets/classification tasks are addressed using deep networks. Similar architectures are used for both tasks. Below we discuss the datasets and their natures, the relevant preprocessing, and the results.

2 Repo Structure

2.1 The Datasets

Both the datasets are available on kaggle.

Most of the code is exclusively contained in jupyter notebooks which also serve as walkthroughs of the implementation. There is one for the rotten tomatoes sentiment analysis dataset, and one for the fake news dataset. The RT sentiments dataset consists of 156,060 phrases and sentiment numbers as well as some other less important labels such as "SentenceId" that can be ignored. The phrases are pieces of a sentence, generated from the original with replacement, which range from one word to most of the sentence. Importantly, for a given sentence the sentiment score can be very different even for phrases from the same sentence. The sentiments are scores taking values in $\{0, 1, 2, 3, 4\}$. The distribution of the sentiments is concentrated at 2, corresponding to a "3 out of 5" or "neutral" rating, with over 50% of the sentiments being viewed as neutral. Many of the phrases consist solely of so called "stop words", such as but, it, this, etc., which are commonly deleted from training sets used for NLP tasks before embedding. However, in this dataset, this would completely delete many of the phrases and result in a large loss in data. We can demonstrate quantified losses from deleting stopwords in terms of accuracy / precision etc., as well as model stability (see the basic ML approach for this problem). In general, not much preprocessing is done for the phrases.

The Fake News dataset consists of 23,196 titles and their classification into real or fake, as well as their url (some missing), domain name, and number of retweets. One can see an exploration of the dataset here: News Data Preprocessing. For the purposes of classification everything but the title will be ignored. The classes are also not balanced, with only 5,755 of the datapoints being fake. As compared to the sentiments dataset, the fake news data titles are preprocessed more heavily. In addition to the needed vectorization and padding, stopwords are removed and the titles are stemmed using the Snowball stemmer. This is more standard practice especially for classifying longer pieces of text. The fake news dataset is much smaller than one may want for a training a deep network, and we will see that while the deep learning approach does well it doesn't do incredibly.

2.2 Code

Two different implementations are included - both in heavily annotated jupyter notebooks. There is also a script that takes a lot of user input to allow one to specify whether to remove stopwords etc. or not

and whether to use the model with or without the convolutional layer. The models will train very slowly on most local machines. Most of the training was done using kaggle GPU access. The reader will likely find the notebooks easier to learn from and these include a more intuitive walkthrough. The script is a satisfying end product though.

3 General Approach and Architectures

The basic architecture consists of

1. An embedding layer of size equal to the number of different words. A dropout layer is added to this embedding to help avoid overfitting.

2. A bidirectional GRU layer, which encodes the main recurrence for the network, providing a sort of "memory" for the network.

* (Optionally) A 1D convolutional layer followed by a pooling layer (no significant difference observed between max and average pooling).

3. A dense layer with 128 units. Another dropout layer is added to this layer.

4. A classification layer.

See figures 1 and 2 for more details on the main architectures. The convolution kernel size, number of hidden dense units were experimented with as part of tuning the hyperparameters.

The main recurrent structure is through the bidirectional GRU layer. GRU stands for gated recurrent unit, and serves as a mechanism for storing features. The gating acts to forget or input certain features based on the previous hidden state. The bidirectional layer (which takes the GRU layer as input), consists of neurons in two directions, one for forward states and one for backwards states. By using two time directions, input information from both the past and future of the current time frame can be used which gives the layer access to more input information. Both these structures deal with looking forwards and backwards in the inputs as a mechanism to incorporate context into the layer's output. Recurrent structures are thus useful in natural language processing tasks because of the contextual nature of language, with a words occurrence before or after another being useful information for classification.

More complex architectures were tried, but they did not improve validation accuracies during training. This included some parallelized architectures with more bidirectional layers. The lack of improvements

Layer (type)	Output Shape	Param #
embedding_17 (Embedding)	(None, 52, 300)	4,586,700
spatial_dropout1d_17 (SpatialDropout1D)	(None, 52, 300)	0
bidirectional_17 (Bidirectional)	(None, 256)	330,240
dense_38 (Dense)	(None, 128)	32,896
dropout_17 (Dropout)	(None, 128)	0
dense_39 (Dense)	(None, 5)	645

Total params: 14,851,445 (56.65 MB)
Trainable params: 4,950,481 (18.88 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 9,900,964 (37.77 MB)

Figure 1: Architecture without Convolutional Layer

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	(None, 52, 300)	4,586,700
spatial_dropout1d_15 (SpatialDropout1D)	(None, 52, 300)	0
bidirectional_15 (Bidirectional)	(None, 52, 256)	330,240
conv1d_15 (Conv1D)	(None, 50, 32)	24,608
global_average_pooling1d_9 (GlobalAveragePooling1D)	(None, 32)	0
dense_34 (Dense)	(None, 128)	4,224
dropout_15 (Dropout)	(None, 128)	0
dense_35 (Dense)	(None, 5)	645

Total params: 14,839,253 (56.61 MB)
Trainable params: 4,946,417 (18.87 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 9,892,836 (37.74 MB)

Figure 2: Architecture with Convolutional Layer

from additional layers is consistent with the discussion in the forum post linked in sources wherein many advocate for fewer hidden layers for most tasks. This discussion is over 13 years old, but is still interesting

to see.

Since the more successful entries in the kaggle sentiment analysis competition utilized the DeBERTa network, I wanted to experiment with a pre-trained network after initially studying the architectures above without a pretrained embedding layer. The approach taken is to use an embedding layer where the weights are set to pre-trained word embeddings, taken from the 2 million word 300 dimensional fast text embeddings. These embeddings are available on kaggle as well.

4 Results and Discussion

4.1 Sentiment Analysis

Since it came from a competition, we have a submission test set to use to additional benchmarking on top of a standard train/test split. The accuracy on the submission set for the pre-trained network was 67%, while the validation accuracy was 69% in the best cases. Without the pre-trained embedding, the test accuracy was about 65% and the validation accuracy about 68%. The submission accuracy for the pre-trained layer is thus a little closer to that on the validation set (about 69% - 67%) compared to the model without a pre-trained embedding layer. This means the pre-trained model seems to generalize a bit better, albeit only 1% better, but an important point is that the early stopping parameter for the pre-trained layer was set to tolerate more increases in validation loss whereas without the pre-trained layer the early stopping was set to tolerate only 1 instance of increased val loss. Changing this patience to be more than 1 for the model without a pre-trained embedding we would see a progressively broader overfitting split in the training/validation accuracies, and less generalization in all likelihood, meaning that the pre-trained network (1) learns slower (which one can see in the notebook) and (2) can be trained for more epochs whilst exhibiting less overfitting and better generalization.

An important technique here is adding a number of model results together, which improves performance by about 0.5%. This is not a huge margin of improvement, but more importantly this helps our model's generalizability and predictive stability.

Viewing the network output as a matrix of probability distributions for the probabilities of each phrase being in the 5 different sentiment classes. The Law of Large Numbers says that sampling the sum of identically distributed random variables will return the mean of the random variables with high probability. For each phrase, we can view the probability of the phrase being in sentiment class X, where X is one

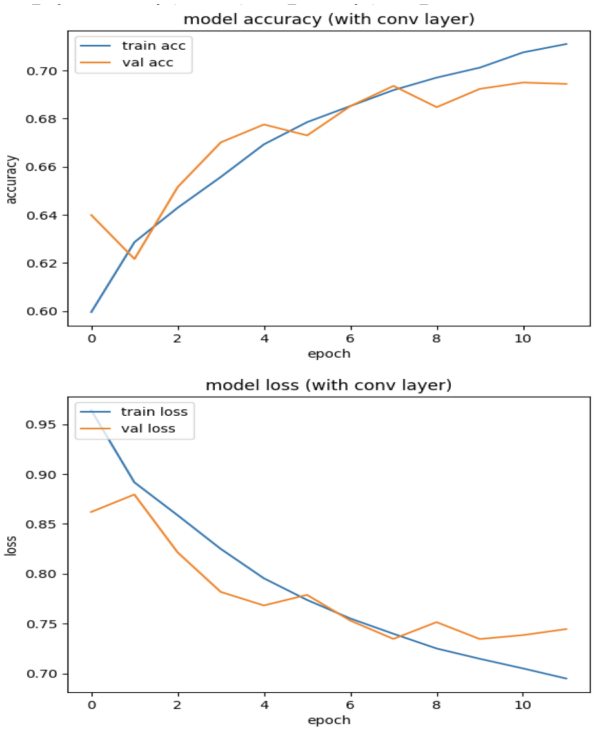

Figure 3: Validation Acc/Loss with Convolutional Layer

of 0,1,2,3,4, as a random variable whose distribution is identical across all the models. Summing the predictions gets us closer to the true class of the phrase X because the sums of the class random variables will be concentrated near their means. In effect, this helps account for any instability in the model and gets rid of some of the noise in the predictions, leaving us with more consistent results from each model.

The performance on the sentiment analysis dataset was good considering the difficult nature of the classification task. Looking at the competition leaderboard, the two best results achieved 76% test accuracy, but the rest of the top 10 submissions ranged from 67.5% to 70%, so the more realistic target is likely this range. Improvements likely come from utilizing DeBERTa or other large scale transfer learning techniques.

4.2 Fake News

The fake news dataset does not have a submission test set, so we report the validation accuracies. For the network without a pre-trained embedding layer, the model achieved validation accuracy of 83%. With pretrained weights, the model achieved 84% validation accuracy.

The confusion matrices illustrate that the performance on the "0" class (fake), which has fewer samples, was similar for both models whilst the pre-trained network performs better on the "1" (real) class. The imbalanced classes may be to blame for this. Model predictions were also added together in a similar

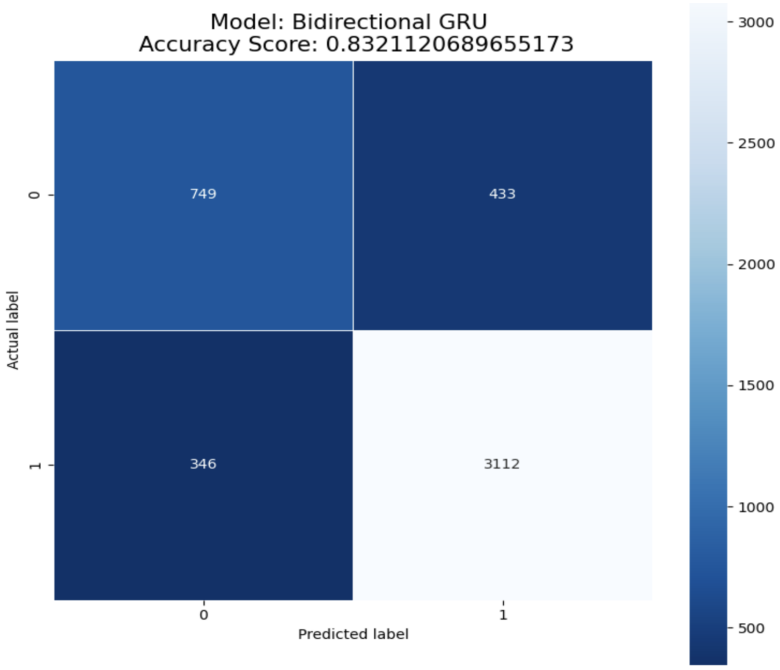

Figure 4: Fake News Confusion Matrix

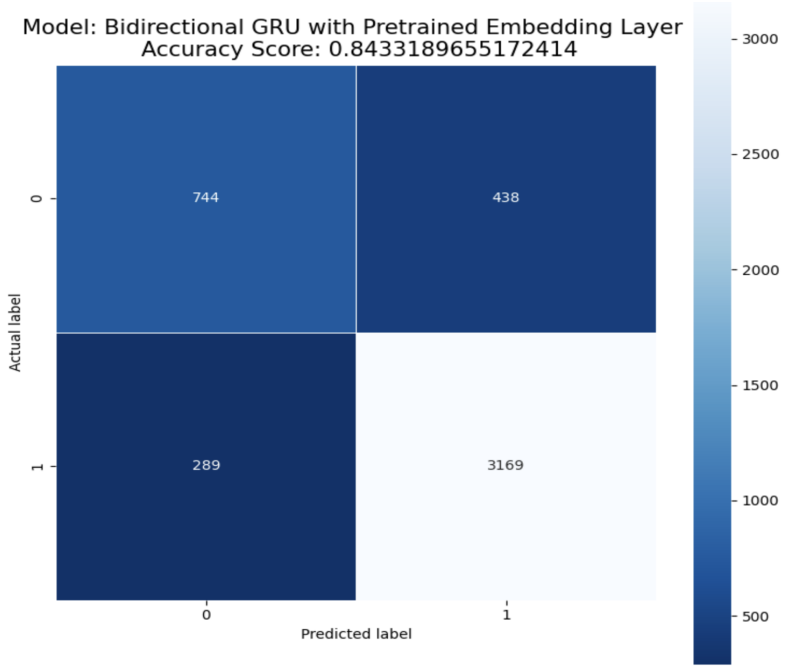

Figure 5: Fake News Confusion Matrix (with Pre-trained Embedding Layer)

manner as in the sentiment analysis implementation, providing a similar 0.5% gain.

The performance of this model on the Fake News dataset was fairly strong given the relatively small number of samples and the imbalanced nature of the dataset. Improving the accuracy here would probably require transfer learning through a larger and very well-trained network.

A Sources

- For more on NLP and word bagging. "Bag of Words Meets Bag of Popcorn" Kaggle tutorial
- For more on stemming, IBM 'What is Stemming?'
- For the original Kaggle competition:
  Addison Howard, Phil Culliton, Will Cukierski. (2018). Movie Review Sentiment Analysis (Kernels Only). Kaggle Link
- Guide to using pre-trained embeddings (how I wrote knew to write the appropriate functions etc.) : Keras Word Embeddings Tutorial
- Forum Discussion on choosing the number of hidden layers in a NN. How to choose the number of hidden layers and nodes in a feedforward neural network?
- Some inspiration for the model architecture was taken from this work, although it was trimmed down significantly. https://www.kaggle.com/code/artgor/toxicity-eda-logreg-and-nn-interpretation