OffensEval Challenge

Fraser Price, Jinsung Ha

Introduction

The OffensEval task requires solutions to three distinct sequence classification tasks; these are classification of (a) offensive text (yes/no), (b) offensive text type (targeted/untargeted), and targeted offensive text target (group/individual/other). The supplied dataset was crowdsourced from twitter, and contains largely political tweets which vary in length- some examples contain only a few words, whereas others contain a number of sentences. This variance in text length, along with the requirement for deep understanding of the semantics, discourse, and pragmatics of the text to achieve high accuracy in all parts makes this a challenging task from an NLP standpoint.

Model Design and Rationale

We believe that every classification task in this challenge fundamentally requires a good understanding of context to achieve good results; an explicit but informative demonstration of this is the use of swear words throughout our dataset. They are often used in colloquial language as exclamations, but are also used as targeted insults- being able to tell the difference between the two requires knowledge of text pragmatics, which in turn requires context. We also theorize that this context should span the entire body of text, as longer tweets may cover a number of related but different topics of discussion over multiple sentences. We therefore ideally require a model which can take the entire text of an example as input, in a form which preserves context, and provide classification as output; this requirement rules out models with non-contextualized features (e.g. bag of words, n-gram embeddings, etc.), and leaves those which are built on top of a language model. For our classifier to perform well on our relatively small corpus, we will also need this language model to be pre-trained. This is because 13,000 tweets is simply not enough data to train a language model from scratch due to the shortage of both words and contextual examples.

Fortunately, recent breakthroughs in NLP have shown that models trained on huge corpuses can be quickly fine-tuned to produce impressively accurate contextual word embeddings, which can then be used as features for a classifier. In particular, a recent study by Google research proposed a technique known as BERT. This involves a language model which is pre-trained on a huge amount of data, including the entirety of Wikipedia, and has shown state of the art results when fine tuned for a vast array of NLP tasks. It is relevant to us because it has been shown that this language model can be fine-tuned in conjunction with an extra feed-forward classification layer in order to rapidly solve end to end sequence classification tasks.

BERT was considered a massive breakthrough in NLP, and builds on a number of previous ideas such as ELMo, ULM-FiT and the OpenAI transformer. It is exciting because developers are now able to trivially build accurate classification models based upon relatively small corpuses using the features a pre-trained BERT model provides.

The BERT model takes a vector of tokenized text as input, and returns contextualized 768-dimensional embeddings for each token. It does this through the sequential stacking of a number of encoder layers (12 in the "base" BERT model we will be using, and 24 in a larger scale version), each one being a bi-directional version of the Transformer proposed by OpenAI which uses an attention mechanism. These stacked layers are trained through a process known as "masking", where a token in the input is hidden, and the network is asked to predict what that missing token is; in this way, the network preserves both forward and backward context during training.

Once we have our contextual embeddings produced by this pre-trained model, we can simply use a single feedforward layer as a classifier.

Model Selection, Training Decisions and Challenges

A pre-trained PyTorch implementation of BERT is publicly available, and also includes optimized code for classification, along with a specialized version of the Adam optimizer which is designed for the fine-tuning of the network. This abstracted away a lot of the design work for us in terms of model design, specifically in the form of architecture, regularization and activation/loss functions.
The GitHub repository for our submission is available at https://github.com/fraserprice/OffensEval, and also contains our trained models.

We will now list the decisions we explored when selecting and training our model and how we approached each, listing challenges we faces as necessary:

- **Large BERT vs. Base BERT**: BERT comes in two flavours; as the name suggests, Large BERT is a deeper version of Base BERT with double the number of encoder layers (24 from 12). Although Large BERT has been shown to objectively perform better than Base BERT on all tasks explored in the paper, we decided that the vastly increased training time was not worth it for the purpose of this exercise. If we had access to more computing resources, we would have gone for the larger model; however we feel that using the base model is enough to demonstrate the validity of the approach.

- **Pretrain Version**: As a number of pretrained models of BERT exist, we were left with the task of selecting which one to use. The main difference is between "cased" and "uncased" (i.e. with capitalization preserved or all text transformed to lowercase); we found that the uncased version empirically worked better for all models.

- **Data Preprocessing**: Along with the capitalization decision described above, we also considered other preprocessing to apply to our data.
  We decided that changing or removing our text (e.g. removal of stopwords) would remove context information and could be destructive to our model; we therefore decided against it.
  The number of occurrences of classes in our data are quite unbalanced across all three tasks; this initially caused issues where our network would simply memorize the most common class and predict this every time without any generalization. We solved this by showing the network an equal, random sample of each class at train time. Note that this was not the case for validation and testing, where the original proportions were preserved.

- **Learning Rate**: The learning rate of the BertAdam optimizer was recommended to be 5e-5; we found that this learning rate worked well, and that changing it had a destructive impact on performance.

- **Batch Size**: For the fastest training possible, we used the largest batch size our GPU would allow; this turned out to be 16, and had better results than lower batch sizes.

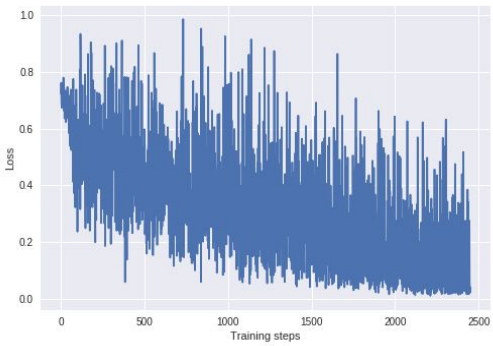
- **Early Stopping**: We handled overfitting by checkpointing our model and validating its performance once per epoch; in this way, we were able to stop training if our validation performance began to diverge from test performance.

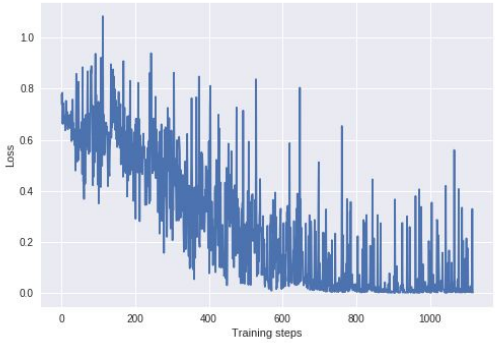
Results

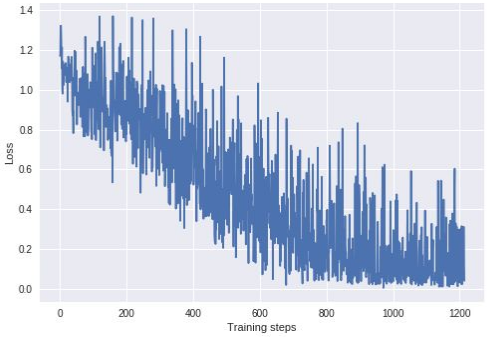
Overall, we believe the performance of our network was very promising. We performed our training on the Google colab platform, and stopped training each task after 4 epochs. We saw signs of overfitting in none of the tasks during this time, but convergence rate slowed down fairly rapidly after around 2 epochs for each model. We believe a higher accuracy may have been possible with more training, but due to the limited runtime on the colab system we decided that 4 epochs was necessary to demonstrate our results. This was a similar decision to the one discussed previously regarding the choice of base BERT over the large version.

Below is a list of loss curves as well as F1 and accuracy scores for each challenge; these scores were calculated using a 10% size test set which was not used in training.

Task A, B, C Loss curves (counterclockwise)







Task A: F1: 0.504, Acc: 0.566
Task B: F1: 0.496, Acc: 0.823
Task C: F1: 0.302, Acc: 0.472

It is evident that our models perform better on some tasks than others (i.e. lower F1 on task 3). We believe this is due to a number of factors to do with the dataset and task rather than our model; specifically, task 3 has a much smaller dataset with just 3500 examples (compared to over 13,000 for task A) while also being a multiclass classification problem as opposed to the binary problem in tasks A and B. We think that for significant improvements to be made on any of the tasks, we would require one of the following:

- Larger dataset for fine tuning, preferably with balanced examples
- More train time on large bert or some other, more accurate language model system
- Ensemble model, possibly using a number of different language models or embeddings

We believe that overall our results show that our single approach is appropriate and robust across the three tasks. This is important not only because the models produced have reasonable accuracy, but also demonstrates the theory that the same technique may be applied to arbitrary sequence classification problems with similar size corpuses.