Sentiment Analysis of Tweets using Deep Neural Architectures

Michael Cai
`mcai88@stanford.edu`

Abstract

The task of sentiment classification of tweets is notoriously difficult due to both the brevity of the form and the common use of nonstandard spellings or slang terms. While many studies have described sentiment classification systems with incredibly high levels of accuracy, most have not been tested on Twitter data. This project describes my experiments with very-deep convolutional neural networks (VD-CNNs) and Google's BERT architecture for classifying tweets in the Sentiment140 data set as positive or negative, which ultimately led to the construction of a model that achieved an F1 score of 0.853 on the included test set.

1 Introduction

Two-way sentiment analysis is a task that many machine learning systems have generally performed very well on. With recent advances in deep neural architectures, it is not uncommon to see ML systems achieve over 95% accuracy when classifying text such as IMDB movie reviews or Amazon.com product reviews. However, few of these systems (at least that I found) were extensively trained and tested on data from Twitter, a social media platform where users communicate through 140 character posts called "tweets". Tweets pose an interesting natural language processing challenge because they can be very difficult to classify due to their limited length, and because Twitter users often employ unconventional spelling and grammar to convey their messages.

In this project, in order to identify positive and negative tweets, I experimented with very deep convolutional neural networks (VD-CNNs), which have been shown to perform very well on sentiment analysis tasks. Specifically, the architecture that I explored is based on the architecture described by Conneau, et. al. that utilizes a convolutional block architecture (described below) to learn the contextual relationships between words in a text. Addtionally, I compared the results of the models I built with results obtained by fine-tuning Google's pretrained BERT architecture, which has generally performed at a state-of-the-art level for most NLP tasks.

Ultimately, this project is both an exploration of the abilities and limitations of deep convolutional neural architectures, and an exercise at applying modern neural network techniques to an established and relevant problem.

2 Related Work

While many techniques for sentiment analysis have been proposed over the years, few have been extensively tested on Twitter data. In this section, we will explore some previous work that has been done on these topics.

2.1 Sentiment Analysis Models

The task of sentiment analysis is one of the oldest and most common tasks in natural language processing. Numerous methods have been tested for solving this problem, such as Naive Bayes and SVMs. However, as with most other machine learning tasks, the past few years has seen a shift away from traditional machine learning methods in favor of more powerful neural network architectures.

Prior to the introduction of BERT, which can be fine tuned for most NLP tasks, in 2018 (Devlin et al. [2018]), state of the art performance on most sentiment analysis tasks was achieved through VD-CNN architectures first introduced by Facebook Research's Conneau, et. al. (Conneau et al. [2017]) in 2017. Through the use of a multi-layered CNN and character level word embeddings, the Facebook team was able to classify the polarity of longer form text - Yelp and Amazon reviews - with 95.72% accuracy. However, their model was not tested on data from Twitter. Like was mentioned above, the model I built and tested for this project was based heavily on the architecture proposed by Conneau, et. al., with the primary difference being that my model utilized pretrained word embeddings from the GloVe corpus instead of character-level embeddings (which I unfortunately did not have enough time to implement in my model).

2.2 Twitter Sentiment

In my research, I found a few attempts at classifying Twitter sentiment using various methods. In their exploration of SVMs for sentiment analysis, Mohammad, et. al. collected tweets labeled positive, negative, and neutral from several sources including the Sentiment140 database referenced in this project (Mohammad et al. [2013]). The group extracted features from each tweet such as character n-grams, number of hashtags, emoticons, etc. for their classification and achieved a F1 score of 69.02 in their 3-way classification. Similar work was performed by Barnes, et. al. in their assessment of text classification techniques. The group trained multiple models on the SemEval dataset for the 3-way classification of Tweets, and achieved the best results using a Bi-LSTM model, with an F1 score of 68.5 (Barnes et al. [2017]). Unfortunately, I was unable to find any literature on the exact task that I am working on for this project, which makes the task all the more interesting.

3 Approach

In this section, we will describe the design decisions we made and the different architectures we explored.

3.1 Model Basics

In order to standardize Twitter data, each tweet in our dataset was either padded using the '<pad>' token or truncated to an arbitrarily set length of 20 words. With the exception of BERT, the models were trained using pretrained 100-dimensional word-embeddings downloaded from the GloVe word embeddings dataset (either the Wikipedia + Gigaword 5 6B or Twitter 27B corpuses). Each tweet in our dataset was given a score of 1.0 for 'positive' and 0.0 for 'negative', so all final results were calculated by using a sigmoid function and then rounding to the nearest value. All models were optimized using an Adam optimizer, with loss calculated via Binary Cross Entropy Loss, and evaluated by comparing both binary accuracy and f1 scores.

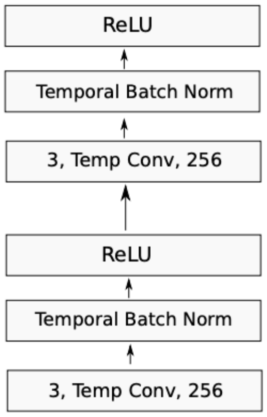
3.2 Baseline Model

For a baseline proof-of-concept model, I built a simple CNN sentiment analyzer based on the simple pytorch sentiment analyzer built by Ben Trevett (Trevett [2019]). The CNN was built with four parallel 100-channel 1-D Convolutions of sizes 1, 2, 3, and 4. The outputs of the convolutions are each fed through a ReLU nonlinearity and then Max Pooled. The results are then concatenated and zeroed out via dropout regularization (p = 0.50), and then fed into a fully-connected layer. Due to computational limitations (I did not have Azure set up yet), I trained our model on a subset of the Sentiment140 dataset containing about 100,000 positive and negative tweets, and then evaluated on a dev set of 10,000 tweets. The model was trained using a local CPU over 50 epochs with a batch size of 128.

As expected, this model is not the most accurate model. It strongly overfits the training data, achieving over 95% accuracy on the training data, while maxing out at 65% accuracy on the dev set by the fifth epoch.

3.3 Deep CNN with GloVe Embeddings

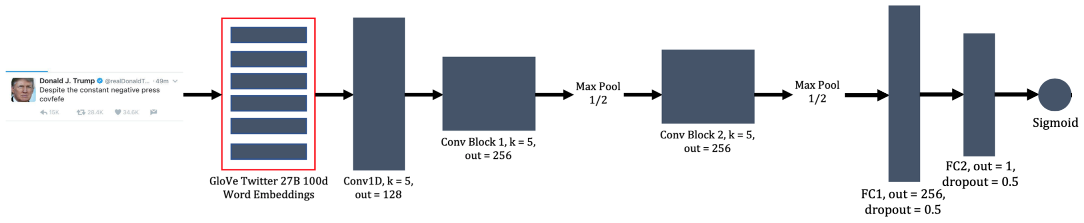
The deep convolutional neural network architecture we constructed was heavily inspired by the work of Conneau, et. al. (Conneau et al. [2017]) The crux of our model is the convolutional block design outlined by Conneau, et. al. which contains two 1D Convolution - Temporal Batch Norm - ReLU sequences, a depiction of which is reproduced below.



This we coded using Pytorch to function as its own discrete class, allowing for maximum reusibility in expanding our neural network depths. For both 1D convolution layers, we used a kernel size of 5, padding of 2, and stride length of 1, and for the Max Pool layer, we used a kernel size of 2. The result of each convolution block is that the number of channels is doubled, and the length of each input sentence is halved.

Like our baseline model, and unlike the model built by Conneau, et. al., our CNN model began with a pretrained GloVe embeddings layer. I experimented with two of the pretrained corpuses offered: the Wikipedia + Gigaword 6B 100d vectors, and the Twitter 27B 100d vectors.

For this project, I tested two different architectures: a shallower network containing two of the aforementioned convolutional blocks (2 Conv Block), and a deeper architecture using four convolutional blocks (4 Conv Block). Both architectures feature two max pooling layers that halve the sentence length, with the 2 Conv Block architectures containing a max pool after each block, and the 4 Conv Block architecture containing one after every other block. In either network, the results of the convolutions is passed through two fully connected layers utilizing dropout regularization with a keep probability of 0.5. Finally, a classification is made using a sigmoid function. Thus our experiments compared the effectiveness of neural networks of various depths with various hyperparameter values for the task of tweet sentiment classification. All infrastructure for this model was built from scratch using functions provided by the Pytorch libraries.


Donald J. Trump @realDonaldT... · 49m
Despite the constant negative press covfefe
GloVe Twitter 27B 100d Word Embeddings
Conv1D, k = 5, out = 128
Conv Block 1, k = 5, out = 256
Max Pool 1/2
Conv Block 2, k = 5, out = 256
Max Pool 1/2
FC1, out = 256, dropout = 0.5
FC2, out = 1, dropout = 0.5
Sigmoid

A 2 Conv Block model with GloVe Twitter 27B 100d word embeddings

3.4 BERT

In addition to the deep CNN models described above, I also performed some experiments fine tuning Google's BERT model using the Sentiment140 dataset. Due to time and computational constraints from using a GPU (instead of a TPU), I was only able to fine tune the model on a subset of our dataset (160,000 examples split between train and dev sets) over only 3 epochs. The model was downloaded from the original Google Research Tensorflow repository on Github and fine tuned using the run_classifier.py script provided in the repository. All the BERT code we used can be found here: `https://github.com/google-research/bert`

4 Experiments

This section will describe the methodologies used to test and tune the models, as well as some of the results obtained.

4.1 Data

We trained our model on the Sentiment140 dataset collected by Stanford graduate students (link: `http://cs.stanford.edu/people/alecmgo/trainingandtestdata.zip`) (Go et al. [2009]). It is free to use and contains about 1.6 million tweets labeled as "positive" (4) or "negative" (0). While documentation online says the dataset contains "neutral" (2) tweets as well, upon observation, we found very few (only about 150) tweets that were labeled as "neutral," so we did not consider them in our model.

For the purposes of our experiments, I adjusted the dataset so that the "negative" tweets were given the label "0" while the positive tweets were given the label "1". The few neutral tweets we did find in the dataset were all given a label of "1" (so in a way, our model is being trained specifically to identify negative tweets). These decisions were made to simplify the tasks of calculating training loss and model accuracy, allowing us to use a simple single-output sigmoid layer to perform the final classification rather than a multi-class softmax layer.

4.2 Evaluation Method

We are currently evaluating our model using two metrics: the binary accuracy score given by the following equation:

$$accuracy = \frac{Correctly\ Labeled\ Examples}{Total\ Examples}$$

and the F1 score given by:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

I chose these metrics because they were the two most common metrics used in literature to judge the effectiveness of sentiment analysis systems.

4.3 Experimental Details

All of the CNN models were trained uisng a batch size of 2048, an adam optimizer, Binary Cross Entropy loss, and gradient clipping for gradients greater than +/- 5.0. Our model ran for up to 100 epochs on an Nvidia Cuda GPU provided by Microsoft Azure. To save time, I employed early stopping once validation results began diverging. Evaluation on the dev set occurred after every epoch (750 iterations). In my experiments, I tuned the model by adjusting the learning rate both manually and through learning rate decay, and also by experimenting with weight decay for L2 regularization. Training the shallower model required about 5 hours, and the deeper model about 7 hours.

The BERT model was trained using the same GPU with a batch size of 256, a learing rate of 2e-5, and, due to time and computational power constraints, 3 epochs. Training and evaluation took approximately 12 hours.

4.4 Experimental Results

The results of my experiments are reported in Table 1 below. For each architecture/hyperparameter combination, I recorded the top F1 score and accuracy achieved on each validation, which usually occurred somewhere between epoch 20 and 30. The top scores overall are **bolded**.

Model	**Dev Set F1 Score**	**Dev Set Accuracy**
Baseline	n/a	0.650
2 Conv Block, Wikipedia 6B, LR = 0.001	0.803	0.797
2 Conv Block, Twitter 27B, LR = 0.001	**0.817**	0.810
2 Conv Block, T 27B, WD = 0.001, LR = 0.001	0.805	0.798
4 Conv Block, T 27B, WD = 0.001, LR = 0.002	**0.817**	**0.811**
4 Conv Block, T 27B, WD = 0.001, LR = 0.0001	0.816	**0.811**
4 Conv Block, T 27B, WD = 0.001, LR Decay	0.812	0.808
4 Conv Block, T 27B, LR Decay	**0.817**	0.810
4 Conv Block, T 27B	0.809	0.807
BERT	n/a	0.808

Table 1: Top F1 scores and accuracies achieved on dev set

The results we found were a little lower than I'd hoped, though just about what I expected. While state-of-the-art models often achieve over 90% accuracy or F1 on most binary sentiment classification tasks, like was mentioned above, the classification of tweets comes with a unique set of challenges. Especially since every model (with the exception of BERT of course) was built from scratch, I think these results are pretty impressive. I was surprised to find that my models actually beat BERT by around 1% for their F1 score, though given more time and computational power, I am sure that BERT would have outperformed any model I built.

The over 1% boost in both F1 and accuracy caused just by utilizing the GloVe Twitter 27B word embeddings instead of the Wikipedia embeddings was right along with what I expected, since any model should benefit from having a corpus that has more tokens and word distributions that more directly match the task at hand.

Interestingly enough, contrary to my preconceptions, adding more layers to the model did not provide any significant boost in model performance. Rather, despite all the different 4 Conv Block architectures I experimented with, the best any model could do was just to match the performance of the 2 Conv Block model. This is in spite of the larger model requiring far more time to train and taking up far more memory due to its large number of parameters. This shows that for twitter binary sentiment analysis, very deep models don't necessarily perform better when compared to relatively more shallow models on this task. This may be because in such a short tweet, there is only so much information one can extract, so a deeper model doesn't enhance performance in the same way it would for a much longer form piece of text. This is actually very similar to a conclusion drawn by Conneau, et. al. where they determined that their VD-CNN architecture performance peaked around a depth of 29 layers (Conneau et al. [2017]).

4.5 Test Set Results

After running these experiments, I determined that I achieved my best results with the 2 Conv Block Twitter 27B model. While the model had a slightly lower dev set accuracy compared to the 4 Conv Block models, the F1 scores were essentially identical, and the 2 Conv Block model has the added benefit of being a much smaller and more lightweight model, and thus easier to train, save, and evaluate. I loaded the cached state_dict for that specific model into a pytorch object and preprocessed the test set with the same data processing techniques as above. On the test set, my model achieved an accuracy of **0.807** and a surprisingly high F1 score of **0.853**.

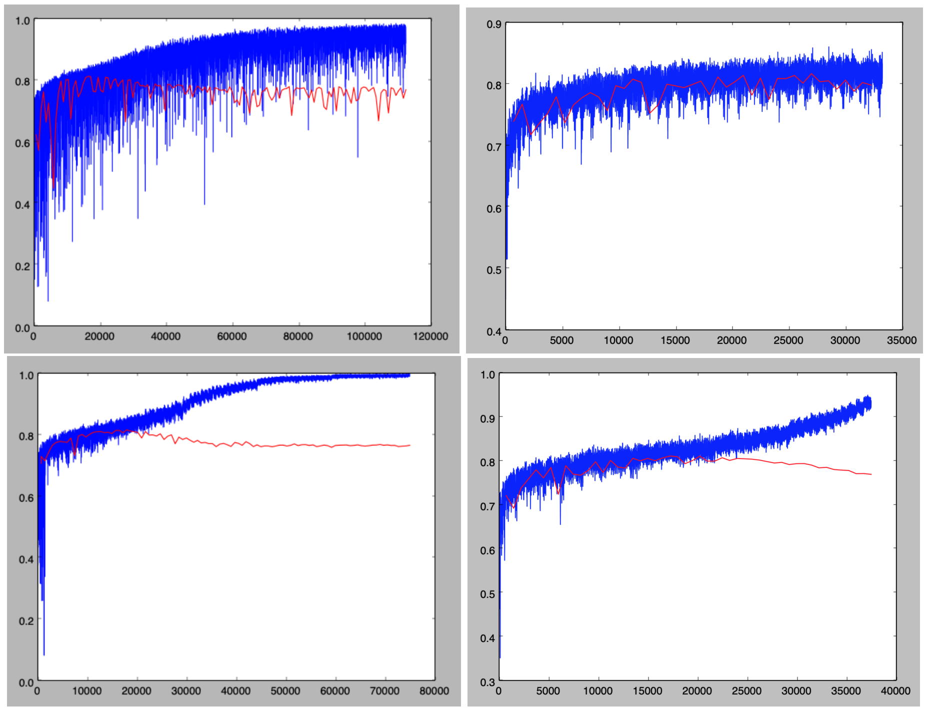
5 Analysis

5.1 Hyperparameter Search

When analyzing my model performance as we train over time, one thing that immediately stands out is that the model tends to very strongly overfit the training data. For example, when comparing the F1 scores on the train and dev sets for the 2 Conv Block Twitter 27B model, we can see that the scores for both train and dev sets increase together to about iteration 20,000, but after that, the train F1 continues to increase while the dev F1 falls again before flattening out around 0.76. Additionally, I noticed that the f1 score tends to swing wildly from one iteration to the next, which is indicative of gradient descent bouncing around the local optima rather than finding the necessary point. As a result of these observations, I attempted to utilize weight decay to reduce overfitting, learning rate decay to better direct gradient descent to the local optima, and a deeper model to better fit the data.

What I found is that weight decay causes the train and dev F1 scores to trend more closely, but that it doesn't seem to improve the model performance on the dev set. Meanwhile, learning rate decay just causes the model to more strongly over fit the train data. When the two techniques are combined, the train and dev scores tend to trend together for a while before the LR decay causes the model to much more strongly overfit the training data to the detriment of the dev set score. Overall, we can see in the following graphs that, while these techniques do seem to reduce noise in the model results, overall, they don't particularly improve the model's performance on the dev set.

Given more time, a more thorough hyperparameter search might have resulted in a good compromise between a tighter fit and closer train and dev set results, but given the experiments I ran, it seems that as of now, the best regularization technique for maximizing dev set score is early stopping.



Clockwise from top left: 2 Conv Block T 27B, 4 Conv Block T 27B with weight decay, 4 Conv Block T 27B with LR Decay, 4 Conv Block T 27B with WD and LR Decay. Training F1 score in blue vs Dev F1 score in red plotted over time (iterations)

5.2 Error Analysis

In order to analyze examples where my model misclassifies tweets, I scanned through the classification results outputted by my model on the test set and compared them with the true classifications of each tweet. I found that often times tweets that are actually positive but contain words with negative connotations (or vice versa) are misclassified. For example:

good news, just had a call from the Visa office, saying everything is fine.....what a relief! I am sick of scams out there! Stealing!

In this example, there are several of words generally associated with very strong negative emotions, especially "sick", "scams", and "stealing". The sentiment from these words likely overpowered the positive sentiment from words like "good news" and "everything is fine", causing an incorrect negative classification. These errors may be fixed in the future by designing an LSTM model with attention to do more to note the relationships between words. A similar mechanism may be at work in this next tweet as well:

@Karoli I firmly believe that Obama/Pelosi have ZERO desire to be civil. It's a charade and a slogan, but they want to destroy conservatism

Here, there are a lot of words with positive connotations, such as civil and Obama. However, the classifier, since it does not recognize case or emphasis, fails to recognize that the most important word in this tweet is the all caps ZERO, which firmly places this tweet in the negative category. Also, this is a relatively long tweet (25 words), and since my model truncated tweets at 20 words, the model never got to see the word "destroy" at the end. Therefore it is entirely likely that by increasing the maximum tweet length from 20 to 28 or 30 could lead to measurable improvement in my model performance, and were I to have more time, this would likely be a good path to explore.

6 Conclusions

In conclusion, deep convolutional neural network architectures can successfully perform binary sentiment classification on Twitter data, though they fail to perform nearly as well at the task as they do other sentiment classification tasks. Interestingly, building deeper models did not result in measurably better model performance, nor did commonly used regularization techniques such as learning rate decay and weight decay. In my experiments, I found that the most reliable model for classifying tweets was the 2 Convolutional Block model utilizing the GloVe Twitter 27B 100d pretrained word embeddings, and early stopping for regularization. This model performed better than an admittedly poorly trained BERT model on the Sentiment140 dev set, and achieved a binary F1 score of 0.853 on the test set, which is pretty good considering the difficulty of classifying tweets. Therefore, in this project, I have shown that deep convolutional neural networks can be an effective tool for the binary classification of twitter sentiment.

7 Additional Information

7.1 Mentor

Pratyaksh Sharma

7.2 External Collaborators

Alex Goodman
CS230 Project Partner
`alexgood@stanford.edu`

7.3 Sharing Project

This project is a joint final project between CS224N and CS230.

References

Jeremy Barnes, Roman Klinger, and Sabine Schulte Im Walde. Assessing state-of-the-art sentiment models on state-of-the-art sentiment datasets. *Proceedings of the 8th Workshop on Computational*

Approaches to Subjectivity, Sentiment and Social Media Analysis, 2017. doi: 10.18653/v1/w17-5202. URL `http://www.aclweb.org/anthology/W17-5202`.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Jan 2017. doi: 10.18653/v1/e17-1104. URL `https://arxiv.org/abs/1606.01781`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. oct 2018. URL `https://arxiv.org/pdf/1810.04805.pdf`.

Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. 2009. URL `https://cs.stanford.edu/people/alecmgo/papers/TwitterDistantSupervision09.pdf`.

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. 2013. URL `http://www.aclweb.org/anthology/S13-2053`.

Ben Trevett. pytorch-sentiment-analysis, Jan 2019. URL `https://github.com/bentrevett/pytorch-sentiment-analysis`.