Twitter Sentiment Analysis with Neural Networks
or
How to Build Neural Networks For Dummies

Pedro M. Sosa, Shayan Sadigh

November 15, 2016

Abstract

1 Introduction: Problem & Motivation

We wanted to learn. At the same time trying to solve the problem of sentiment analysis on twitter data, which could provide useful information for companies/brands.

2 Building A Feed-Forward Neural Network

2.1 Basics

Neurons, Weight, etc.

2.2 Feed Foward Calculation

How we calculate the feed foward

2.3 Backpropagation

How we do the backpropagation

3 Pre-processing Datasets

3.1 Word Embeddings

Word embeddings seem to be a good way of representing words. We chose 32 sized arrays for each word.

3.2 Twitter Specific Features

Word-Embeddings is a pretty generic form of describing any text. However, Twitter has very particular features that distinguish it from other text forms. For example, Twitter data can contain mentions *(@user)* and hashtags *(#topic)* that could provide very meaningful. Furthermore, Tweets are only 140 characters long, which might lead people to shorten words in unexpected ways or make more constant the use of emojis. Knowing this, we decided to use represent tweets as a feature vector.

While our original feature vector contained multiple different features, we used WEKA **(TODO: add specific method)** to narrow the features that proved to be more descriptive. The following table shows all the features tested and which were selected:

Feature Name	Description	Selected
chwd	# of Characters/ # of Words	✓
exclamation	# of exclamation marks (!)	✓
smile	# of positive emoticons	✓
sad	# of negative emoticons	✓
url	# of URLs shared	✓
ellipsis	# of ellipsis (...)	✓
mention	# of mentions (@someone)	✓
netprob	$\sum_{W \in Tweet}(P_{pos}(W) - P_{neg}(W))$	✓
question	# of question marks (?)	-
pronoun	# of pronouns (I, me, mine...)	-
hashtags	# of hashtags (#topic)	-
capitals	# of uppercase letters	-
length	Length of the Tweet	-

Table #1 - List of features tested. The selected one refer to the the ones used selected through Weka and used in our experiments.

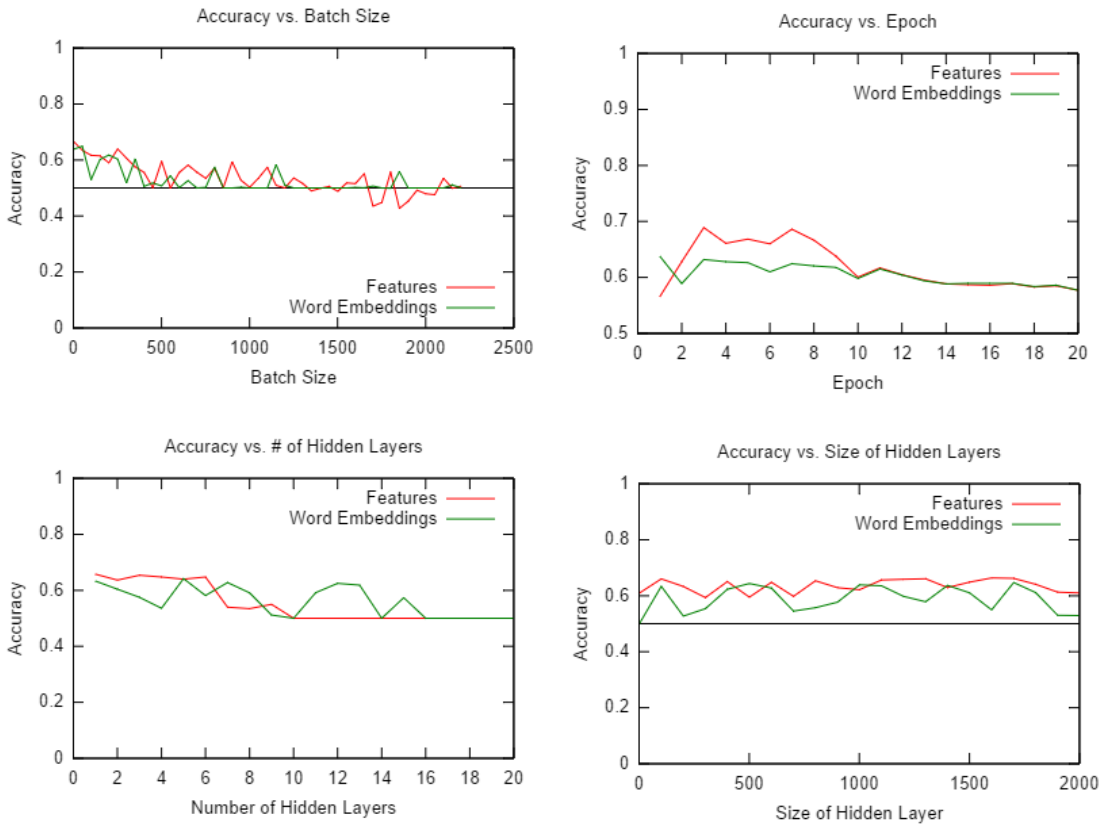

Figure 1:

4 Experiments

Once we had built our own Neural Network and had parsed our dataset into word embeddings and feature vectors we proceeded to test different scenarios to find the best possible configuration. We used a Neural Network built with Keras as the baseline for the following experiments

4.1 Selecting Parameters

Our initial experiments consisted on evaluating the best parameter configuration for a Neural Network. These parameters were: batch size, # of hidden layers, size of hidden layers, and # of epochs.

4.1.1 Batch Sizes

A Neural Network can be trained with batches of multiple training cases at a time. These batches (also known as minibatches) allow users to cut dataset into smaller segments, feed it through the Neural Network, and update the NN with the mean of the final gradient descent.

Having big batches allows the NN to learn from a big dataset faster, as the throughput of each operation is much higher. However, since the update is being done with the mean of the gradient descent of all the batch's training cases, it is possible to loose precision and over-fit to the training data.

Our experiments (Figure 1) clearly show that the NN performed better with batches of around 1 - 300 in size. Higher batches tended to converge to a 0.5 or worse accuracy, which could be attributed to overfitting.

4.1.2 # of Hidden Layers

- As we increase the number of hidden layers, our accuracy decreases. The reason for this is the gradient loss. (talk lots here)

4.1.3 Size of Hidden Layers

While increasing the size of a single hidden layer substantially decreased computational performance, it did not seem to affect the accuracy of the NN. However we did see a rise in dead-paths (i.e neural connections with weight = 0). It is important to mention however, that having a hidden layer that was smaller than the size of the input was actually detrimental to the performance while using Features as an input. This might be because the amount of neural paths are so limited that no real features can be distinguished. For the sake of performance, we that an apporpriate size for a layer was around 250-500.

4.1.4 # of Epochs

The number of Epochs refers to the number of times an NN is trained with the same test data. Having multiple repetitions might be useful especially if you have a small training set, however too many repetitions can quickly lead to over-fitting. In our set of experiments we found that the optimum number of epochs was 2-3. Anything past this value, would slowly tend to overfit and thus produce worse results.

4.2 Word Embeddings vs. Feature Vector

The actual very best setup of Word Embeddings vs. Feature Vectore in Keras

4.3 Keras vs. Our Own Neural Network

5 Further Work

5.1 Convolutional Networks

5.2 LSTM