CNNs for sentiment analysis

Ti Liang, Clay Riley, Xinhao Wang

Responsibilities

Ti Liang
Designed and converted the TensorFlow word2vec and CNN scripts for use with one another and with the Yelp dataset.

Clay Riley
Worked on developing pre-training word embeddings

Xinhao Wang
Ran various models in order to optimize parameters

All members
Write-up and background knowledge

Method and results

Materials
We used Google's TensorFlow v1.1.0 library and the scikit-learn v0.18.1 library in Python to construct the neural networks used in this work. Both multilayer perceptrons (MLP) and convolutional neural networks (CNN) were used, as was Google's basic word2vec algorithm for embedding words into a dense vector by predicting neighboring tokens from a given one (skip-gram).

Google's pre-trained 300-dimensional word embeddings were used as a baseline for our own embeddings tuned to the dataset. These are derived from an English language news corpus using the word2vec (skip-gram). Each dimension is in range (-1, 1).

The Yelp sentiment dataset was used for refining the embeddings and then conducting the experiments. This 330,000-instance corpus consists of free text reviews of restaurants and a 3-way classification of "positive", "negative" or "neutral". Vectors for word types not appearing in the Google News embeddings were randomly initialized and then tuned alongside the pre-trained ones.

The dataset was preprocessed using NLTK's sentence- and word-tokenization functions. NLTK's default English stopwords were removed from text sequences before tuning embeddings, training or testing.

Classifiers

In previous work, a maximum entropy model was run on the dataset, but using sparse vectors to represent vocabulary.

Here, two small MLPs were run to get a baseline. They consisted of two hidden layers of 50 and 3 nodes respectively, trained with a batch size of 200 with rectified linear units (ReLU) as the activation functions. Dropout was not performed in the MLP, and the L2 regularization parameter was set to 1e-5.

For one, a subset of the data was used (N=10,000) and no pre-training was performed. Instead, all vocabulary items were randomly initialized.

Batch size refers to the number of training instances used for each round of backpropagation; error is not calculated until each batch is complete. Smaller batches result in more frequent and more randomly distributed training, which decreases overfitting but increases time spent computing values for the backpropagation process. ReLU converts the output of a given hidden unit *x* to max(0, *x*) in order to introduce a source of non-linearity to the network in a way that maximizes the values found. L2 is another way to prevent overfitting by normalizing error by the sum of the squared weights feeding into a node. Stronger L2 may slow down training progress.

Our main classifier is CNN. Its general architecture consists of 1 constant word embedding layer, followed by 1 convolutional layer, followed by 1 max pooling layer, followed by 1 dropout layer, followed by 1 output layer. Output layer has 3 nodes, one for each sentiment, and the number of nodes of the rest layers are determined by the filter numbers and the max document length fed into the input layer.

We used several different filter windows in the convolutional layer. This maps to convolutions over bi-, tri-, 4-... grams. As for feature maps, we use a constant number 128 for each filter. 0-padding is also used to make input tensor shape compatible with the initial layer. All convolutional layers have an activation function of RELU. After that we do max pooling over the full tensor, which gives a single maximum unit for each feature map of each filter.

The layers after the max pooling connects and flattens all previous filters and propagates them into the following layers that are quite identical to the general structure of an MLP. Dropout is implemented to prevent overfitting. It turns inactivates each node with a certain probability. In our experiment, the dropout has a probability of 0.5 vs. no dropout. Both strong and weak L2 regularizations are tried in CNN.

Results

Model	Batch size	Filter size	L2	Dropout	N hidden	Accuracy	Data size
MaxEnt	n/a	n/a	n/a	n/a	n/a	0.67	10,000
MLP	200	n/a	1e-5	1	50, 3	0.733	330,000
MLP	200	n/a	1e-5	1	50, 3	0.704	10,000
CNN	64	3,4,5	0	0.5	384	0.761	330,000
CNN	64	3,4,5	1e-2	0.5	384	0.757	330,000
CNN	64	2,3,4,5,6	0	0.5	640	0.759	330,000
CNN	64	3,4,5	0	1	384	0.761	330,000
CNN	1024	3,4,5	0	0.5	384	0.76	330,000
CNN	1024	2,3,4,5,6	0	0.5	640	0.762	330,000
CNN	64	3,4,5	0	1	384	**0.764**	10,000

Future Observation:

Unbalanced Dataset
Since Yelp data set is an unbalanced dataset (Positive 60%), if we can sample from the category that has more review, it may help.

Unknown Word Issue
In our experiment, our vocabulary size 200,000, only 28,000 words are in Google Word Vector, which has a vocabulary of 3,000,000. That means most of the word are not common. Some of them may be Name Entity, but some are not. Here are some examples: ‘niice’, ‘preeeeeeeeety’. If we can train another model or a spelling corrector to deal with this problem and turn those word into words in Google Vector, performance can be improved.

Dataset size
We found that unexpectedly, the smaller dataset resulted in competitive accuracy scores. This could be a result of a mismatch between the pre-trained vectors and the high number of vocabulary items not covered by them.

Codebase

Dataset: Yelp review data (https://www.yelp.com/dataset_challenge)

Github repo (2 branches): https://github.com/signalw/cnn-text-classification-tf
https://github.com/tensorflow/tensorflow/blob/r1.1/tensorflow/examples/tutorials/word2vec/word2vec_basic.py
CS server: /home/g/grad/xinhao/IE_Project
Google News word vectors: https://code.google.com/archive/p/word2vec/