

Deep Learning for Sentiment Analysis

Term Project Report

Chunxu Tang
ctang02@syr.edu

Zhi Xing
zxing01@syr.edu

1 Introduction

Due to the “world of mouth” phenomenon, mining the social media has become one of the most important tasks in Data Mining. Particularly, Sentiment Analysis on social media is useful for various practical purposes such as brand monitoring, stock prediction, etc. Sentiment Analysis is inherently difficult because of things like negation, sarcasm, etc. in texts, but Machine Learning techniques are able to produce accuracy above 90% for multi-class classification in regular texts such as movie reviews, arguably better than human. Unfortunately, the irregularities of social-media texts, such as misspelling, informal acronyms, emoticons, etc., make social-media-oriented Sentiment Analysis, or Text Mining in general, extremely difficult.

The buzzing Deep Learning is dominating pattern recognition in computer vision and voice recognition. As it turned out, it may be good at text classification as well. Various deep neural nets achieve state-of-art sentiment-polarity classification on Twitter data (about 87%) [2, 3, 5]. One of the advantages of Deep Learning is its ability to automatically learn features from data, and this ability leads to lots of interesting designs [1, 2, 3, 4, 5].

The goal of our term project is to get a good understanding of Deep Learning techniques and apply it to social-media sentiment classification. We'll first *study* the literature and online articles/tutorials to understand different types of deep neural nets, then apply one or two of them on two-class sentiment classification of Twitter data. We envision the following *programming* tasks:

- Data collection (Chunxu)
- Logistic Regression with Word Embeddings (Chunxu)
- Recurrent Neural Network with Word Embeddings (Chunxu) *
- Convolutional Neural Network with Word Embeddings (Zhi)
- Dynamic Convolutional Neural Network with Word Embeddings (Zhi) *

2 Background

2.1 Word Embedding

2.2 Convolutional Neural Network (CNN)

Deep Learning is pushing the cutting-edge of computer vision, and one of the essential reasons is Convolutional Neural Network (CNN). The key characteristic of CNN that makes it so successful

*These will be done if time permits.

is its ability to automatically select features from inputs. The convolutional layer of a CNN acts like a sliding window over the inputs. At each step in the sliding, normally referred to as a *stride*, the convolutional layer reduces the set of inputs within the window to a single output value. This transformation is done at every stride, and at the end, the input is mapped to a smaller set of output. This is not special for neural network. What is special, however, is that the same convolutional layer is applied repeatedly to all the inputs, and therefore there's a much smaller set of parameters need to be learnt, which is why the network can be "deep". As an example, consider a training set of 100×100 pictures, a convolutional layer may have a window size of 10×10 , so it takes the input 10×10 values at each stride and moves from left to right, top to bottom, converting the 100×100 picture to a much smaller one. The actual resulting size depends on the *stride size*, which is the number of pixels the window slides.

The set of parameters, once learnt, make the convolutional layer specialize at a certain aspect of the input. In a typical CNN, there're a number of parallel convolutional layers, called *channels*, each of which specializes at a different aspect []. These aspects are the "features" learnt by the CNN. In the field of computer vision, one convolutional channel may specialize in detecting horizontal edges, while another may specialize in detecting vertical edges; one channel may specialize in colors, while another may specialize in contrasts. In the context of text mining, the 2-D picture becomes 2-D representation of sentence, which is normally obtained by converting a sentence to a sequence of word embeddings. Since different position in the embedding can be seen as a different aspect of the word's meaning, the hope is that the convolutional layers can specialize so that CNN is able to automatically detect useful features from these very abstract word embeddings.

In a typical CNN, a convolutional layer is normally followed by a pooling layer, e.g. max pooling, which selects the most important the features. There can be multiple repetitions of convolution-pooling pairs and the final pooling layer is normally connected to a fully connected layer, which generates outputs for classification.

3 CNN for Twitter sentiment

3.1 Network structure

The structure of the CNN used in our term project is shown in Figure ???. It is a simplification of the model used in [3]. Although this model has a minimalist design, it has most of the typical layers of a text mining CNN: word embedding layer, convolutional layer, max-pooling layer, and fully-connected layer.

The first layer is the word embeddings, which is encoded as a weight matrix that is used as a lookup table. Each row of the weight matrix is a vector of size k represents a unique word in the vocabulary. Given a sentence of length n , padded if necessary, each word is replaced by its corresponding vector and the sentence is converted to an output matrix that is a concatenation of all the embeddings. This output matrix is represented as

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$$

where $\mathbf{x}_i \in \mathbb{R}^k$ is the embedding of the i -th word in the sentence, and $\mathbf{x}_{i:j} \in \mathbb{R}^{kj}$ is used to denote the sub-matrix from the i -th word to the j -th word.

The second layer is the convolutional layer. Given a window size h , this layer is encoded by a *filter* $\mathbf{w} \in \mathbb{R}^{hk}$. When this filter is applied to the h -gram $\mathbf{x}_{i:i+h-1}$ of the input sentence, a feature c_i is generated by

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

where $b \in \mathbb{R}$ is bias and f is the activation function such as rectifier. The filter slides over all the possible h -grams of the input sentence to produce a feature map

$$\mathbf{c} = (c_1, c_2, \dots, c_{n-h+1})$$

for $\mathbf{c} \in \mathbb{R}^{n-h+1}$.

The next layer is the max-pooling layer. It is applied to the feature map produced by the convolutional layer to produce a single feature $\hat{c} = \max(\mathbf{c})$. Taking the maximum essentially picks the most important feature in the feature map, which is an effective way to deal with variable sentence length, because the special word for padding has 0s in all the dimensions in its word embedding.

The last layer is a fully-connected layer. The \hat{c} is the selected feature by a *single* filter, there're a number of filters with different window sizes. All the selected features from these filters are the input for this fully-connected layer, which uses softmax function to calculate the score for each class.

3.2 Regularization

The purpose of regularization is to prevent overfitting or co-adaptation. Unlike [3], only dropout is used to prevent co-adaptation, because according to [6] the L2-norm constraint used in [3] generally has little effect on the end result, and we want to keep the network as simple as possible. The dropout is applied to the fully-connected layer of the CNN. It works by randomly setting a proportion p of hidden units to 0 during learning. During testing, the dropout is disabled and the learnt weights are scaled down by p .

4 Experiment

4.1 CNN

The `word2vec` model proposed in [] is used. The parameters of `word2vec` model are set untrainable so this layer is essentially used as a mapping from word IDs to word embeddings. There're two reasons the parameters are set fixed:

1. To reduce the number of parameters need to be learnt.
2. Tweets contain lots of noise, making the layer trainable exposes it to the noises, which may be counterproductive.

Because of this layer, the CNN has a vocabulary determined by the `word2vec` model, saved as a word-to-ID dictionary. During data preprocessing, each word is converted to an ID according to the dictionary.

References

- [1] Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.
- [2] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.
- [3] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

- [4] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.
- [5] Xin Wang, Yuanchao Liu, Chengjie Sun, Baoxun Wang, and Xiaolong Wang. Predicting polarities of tweets by composing word embeddings with long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, volume 1, pages 1343–1353, 2015.
- [6] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.