
Twitter Sentiment Analysis with Recursive Neural Networks

Ye Yuan, You Zhou

Department of Computer Science

Stanford University

Stanford, CA 94305

{yy0222, youzhou}@stanford.edu

Abstract

In this paper, we explore the application of Recursive Neural Networks on the sentiment analysis task with tweets. Tweets, being a form of communication that has been largely infused with symbols and short-hands, are especially challenging as a sentiment analysis task. In this project, we experiment with different genres of neural net and analyze how models suit the data set in which the nature of the data and model structures come to play. The neural net structures we experimented include one-hidden-layer Recursive Neural Net (RNN), two-hidden-layer RNN and Recursive Neural Tensor Net (RNTN). Different data filtering layers, such as ReLU, tanh, and drop-out also yields many insights while different combination of them might affect the performance in different ways.

1 Introduction

Sentiment analysis has been a popular topic in the field of machine learning. It is largely applied to data that comes with self-labeled information such as movie reviews on imdb. A scalar score comes along with the review text a user writes, which provides a good and reliable labelling of the text polarity. This ability to identify the positive or negative sentiment behind a piece of text is even more interesting when it comes to social data. Twitter gets new user data literally every second. If our model can predict sentiment labels for incoming live tweets, we'd be able to understand the most recent user attitude towards a variety of topics from a commercial flight satisfaction to brand image. We used a logistic regression baseline model and complex-structured neural networks, Recursive Neural Network(RNN) and Recursive Neural Tensor Network(RNTN). Considering the nature of tweets, we first preprocessed the tweets, built a binary dependence tree as the input to the RNNs. We tuned our hyper-parameters and applied regularization methods such as L2 regularization as dropouts to optimize the performance.

2 Related Word

Researchers have applied traditional machine learning technologies to solve the sentiment analysis problem on the Twitter data set. Agarwal et al [1] proposed a method to incorporate tree structure to help feature engineering. On the other hand, deep learning researchers have a more natural way to train directly on tree structure data using recursive neural networks[2]. Furthermore, complex models such as Matrix-Vector RNN and Recursive Neural Tensor Networks proposed by Socher, Richard, et al.[4] have been proved to have promising performance on sentiment analysis task. This motivates us to apply deep learning methods to the Twitter data.

3 Technical Approach and Models

3.1 Preprocessing

Due to the specific format (for example, 140 character limit) and the mostly casual nature of Tweets, the vocabulary used in Tweets are very different from formal English used in popular NLP datasets such as the Wall Street Journal Dataset. Tweets contains a lot of emoticons, abbreviations and creative ways of expressing excitement such as long tailing (ex. happyyyy). We normalize all letters to lowercase and perform abstractions such as representing any "@USERNAME" as a "<user>" token and convert a single "#hashtag" input to a "<hashtag>" token and a token with the actual tag value "hashtag". Our preprocessing script is based on the Stanford nlp twitter preprocessing script[6].

3.2 Logistic Regression Baseline

First, we establish our baseline model as a simple logistics regression model using the Bag of Word representation. Besides extracting words (unigrams) from the tweets, we also include word bigrams as input features to include introduce some context information to the model. The model is trained with stochastic gradient descent. Here our task is a multi-label classification problem. Our baseline model is a combined model with a positive classifier, a negative classifier and a neutral classifier. To output a final label, the model looks at the three scores produced by the three sub-classifiers and chooses the one label with highest score.

Each tweet is represented by a sparse vector of word counts, denoted by x . Each sub-classifier learns a weight vector w based on training examples minimizing the hinge loss.

$$\text{Loss}_{\text{hinge}}(x, y, w) = \max(0, 1 - w \cdot \phi(x)y)$$

3.3 Recursive Neural Network: Two-Layer RNN and One-Layer RNTN

We used the cross-entropy loss defined as

$$CE(\theta) = - \sum y_i \log \hat{y}_i,$$

where y is the one-hot representation of the actual label and \hat{y} is the probability prediction output by the softmax layer and θ is the set of our model parameters.

3.3.1 Two-Layer RNN

Forward Propagation:

$$\begin{aligned} \hat{y} &= \text{softmax}(\theta) & \theta &= Uh^{(2)} + b^{(s)} \\ h^{(2)} &= \text{ReLU}(z^{(2)}), \text{ where } \text{ReLU}(z) = \max(z, 0) & z^{(2)} &= W^{(2)}h^{(1)} + b^{(1)} \\ h^{(1)} &= \text{ReLU}(z^{(1)}) & z^{(1)} &= W^{(1)} \begin{bmatrix} h_{\text{Left}} \\ h_{\text{Right}} \end{bmatrix} + b^{(1)} \end{aligned}$$

where $h^{(1)} \in R^d$ is either the word vector at leaf node or a function of $h^{(1)}$'s from its children. $h^{(2)} \in R^D$ and $\hat{y} \in R^n$. d is the dimension of word vectors, D is dimension of the hidden layer and n is the dimension of the output layer.

Back Propagation:

For the root node:

$$\begin{aligned} \delta_3 &= \frac{\partial J}{\partial \theta} = \hat{y} - y & \frac{\partial J}{\partial U} &= \delta_3 h^{(1)T} & \frac{\partial J}{\partial b^{(s)}} &= \delta_3 \\ \delta_2^{(2)} &= U^T \delta_3 \circ \text{ReLU}'(z^{(2)}) & \frac{\partial J}{\partial W^{(2)}} &= \delta_2^{(2)} h^{(2)T} & \frac{\partial J}{\partial b^{(2)}} &= \delta_2^{(2)} \\ \delta_2^{(1)} &= W^{(1)T} \delta_2^{(2)} \circ \text{ReLU}'(z^{(1)}) & \frac{\partial J}{\partial W^{(1)}} &= \delta_2^{(1)} \begin{bmatrix} h_{\text{Left}} \\ h_{\text{Right}} \end{bmatrix}^T & \frac{\partial J}{\partial b^{(1)}} &= \delta_2^{(1)} \\ \delta_{\text{below}} &= W^{(1)T} \delta_2^{(1)} \end{aligned}$$

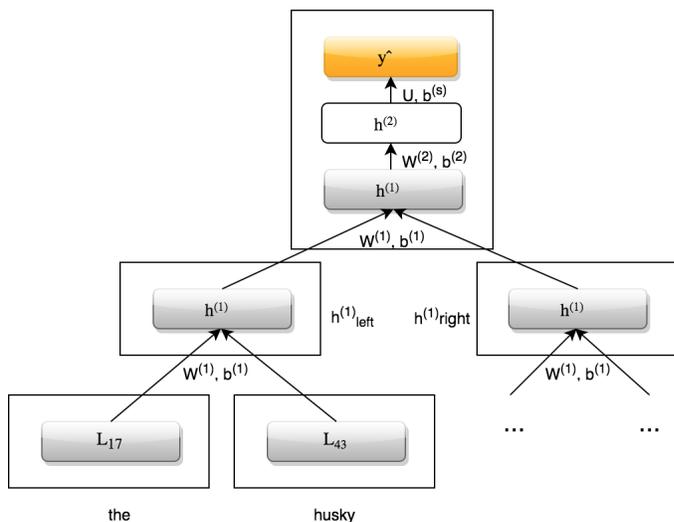


Figure 1: Example Two-Layer Recursive Neural Network Structure

For intermediate nodes:

$$\delta_2^{(1)} = \delta_{above} \circ ReLu'(z^{(1)}) \quad \frac{\partial J}{\partial W^{(1)}} = \delta_2^{(1)} \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix}^T \quad \frac{\partial J}{\partial b^{(1)}} = \delta_2^{(1)}$$

$$\delta_{below} = W^{(1)T} \delta_2^{(1)}$$

Note here δ_{above} refers to either the first half or the second half of the δ_{below} from the higher layer, depending on whether the node is a left or a right child.

3.3.2 One-Layer Recursive Neural Tensor Network

The general structure of the RNTN described by [4] is similar to that of the RNN. We've taken away the hidden layer $h^{(2)}$ and we used \tanh as the activation function for $H^{(1)}$. The important model formulation follows, Forward Propagation:

$$\hat{y} = softmax(Uh^{(1)} + b^{(s)})$$

$$h^{(1)} = \tanh(z^{(1)})$$

$$z_k^{(1)} = \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix}^T V^{(1)[k]} \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix} + W^{(1)} \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix} + b^{(1)}$$

Back Propagation:

$$\frac{\partial J}{\partial V^{(1)[k]}} = \delta_2^{(1)} \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix} \begin{bmatrix} h_{Left} \\ h_{Right} \end{bmatrix}^T$$

Due to space limitation, we omit the other derivatives similar to what we did in section 3.3.1.

4 Experiment

4.1 Data Set

We used the SemEval-2013 data set collected by York University[7], which consists of 6092 rows of training data. We further divided the training data into a training set of size 4874 and a dev set

of size 1218. Examples in the original data set are classified with five labels: negative, objective, neutral, objective-OR-neutral and positive. In fact, the difference between objective-OR-neutral and objective/neutral labels are not very well defined, so for the purpose of our project, we treated the objective class, neutral, and objective-OR-neutral all as neutral examples.

4.2 Evaluation Metric

Naturally, we chose accuracy as our performance metric for this classification task. At the same time, we choose the average F-1 score of the positive and negative groups as our metric so that we have integrate the precision and recall on the two class labels together.

4.3 RNN input format

A recursive neural network requires the training data to have a pre-determined tree structure. We used the PCFG Stanford NLP Parser[3] to build estimates of the actual optimal tree structures. We chose to run the parser basing on a careless probabilistic context-free grammar model, which works better than traditional PCFG models on less strictly grammatical input data such as tweets in our case. Moreover, our recursive neural network assumes each non-leaf node to have two children. So we also binarized our parse tree using a binarizer based on Michael Collin’s English head finder. After these processes, all non-leaf nodes in our parse tree have at most two children. It is possible that a node has only one child, for example $NP \rightarrow N$. We chose to soft delete this node in our NN implementation where cost and errors are directly passed to the next level without modification at this level.

4.4 Regularization

Neural networks are much more powerful than our baseline logistic regression model because they can learn complex intermediate units(neurons) and capture nonlinear interactions between inputs. They are also prone to over-fitting for the same reason. They are so powerful that they usually fit noises in the training data as well as the general model. In order to generalize the model to unseen data sets, we put a lot of emphasis on regularization methods.

First of all, we applied a standard L2 norm on the U and W parameters, as well as the V parameters in RNTN to avoid overfitting. Furthermore, we experimented with the dropout regularization described by Srivastava et al[5]. The idea is to randomly omit half of the neurons at training time for each iteration, which allows us to achieve the same effect as if we are training on $2N$ individual neural networks with N being the number of neurons. We applied dropout to the softmax layer of RNN and RNTN models.

4.5 Results

We initialized our word vectors with GloVe word vectors pre-trained on 2 billion tweets published by the Stanford NLP group. [8]

Experimenting with different combination of layers with neural net models, the optimal combination for each model is:

-	Drop-out	ReLU	Tanh
One-hidden-layer RNN	Yes	Yes	No
Two-hidden-layer RNN	Yes	Yes	No
RNTN	Yes	No	Yes

Hyper parameters also play a significant role affecting the performance. The parameters we have been tuning include:

epochs: epoches number

step: step size

wvecDim: word vector dimension

middleDim: Dimension of the second hidden layer (only applied to RNN2 and RNTN) minibatch:

size of minibatch
rho: regularization strength

For RNN:

RNN best suits this data set among the three net structures. Since labels on the word- and phrase-levels aren't completed, RNN2 and RNTN didn't give much leeway when models fit the data. However, this structure also severely suffers from very fitting, as being such a shallow net structure, fitting all the training data can be challenging. Due to this phenomena, we adjust the regularization force to correct the overfitting, which we can see from Figure 2. The best performance is given at $\text{reg} = 8 \times 10^{-4}$

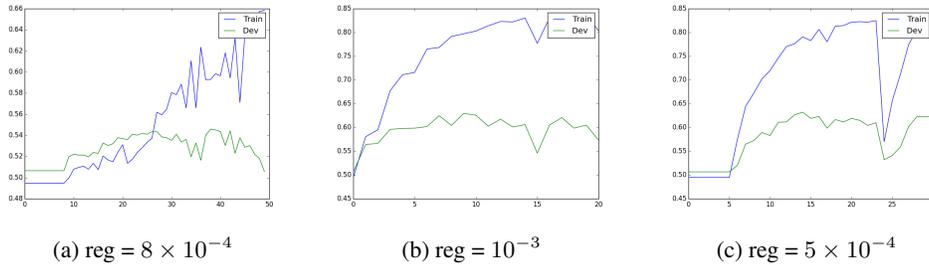


Figure 2: Examples of how regularization strength affects performance in RNN

The confusion matrix of RNN gives us more insight of about the performance. We can see that the model is not good at predicting negative label, due to the lack of negative training data. Barely any instance is classified as negative. It is doing a decent job in neutral and positive labels. The problem also appears in RNN2 and RNTN models, due to the imbalanced training data is used to train all the three models.

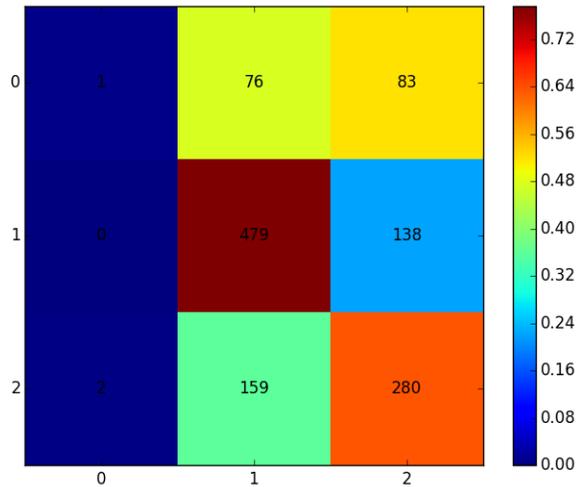


Figure 3: RNN confusion matrix

For two-hidden-layer RNN:

In the two-hidden-layer RNN, over-fitting is not as severe as in the one-hidden-layer RNN, but a more appropriate regularisation strength can still give a rise on the performance, see Figure 4. In RNN2, $\text{reg} = 1 \times 10^{-3}$ gives the best performance. In RNTN, $\text{reg} = 1 \times 10^{-3}$ gives the best performance.

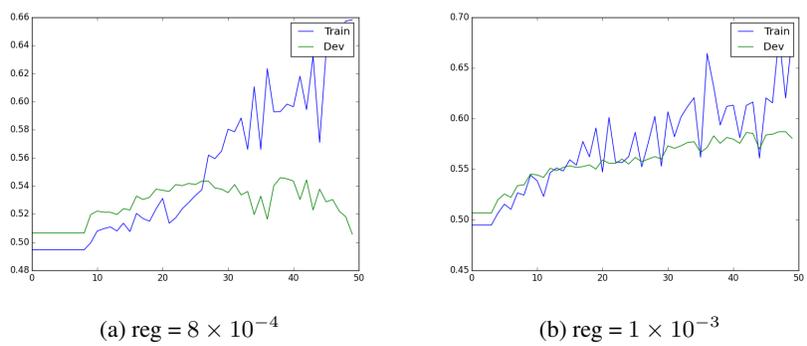


Figure 4: Examples of how regularization strength affects performance in RNN 2

Apart from regularization, the dimension of the middle hidden layer also come to play, since the two-hidden-layer RNN has one more layer than the one-hidden-layer RNN that can be tuned.

We can see that despite the general over-fitting phenomena, when middle dimension is 25, the model gives better performance in terms of data over-fitting and dev accuracy.(Figure 5)

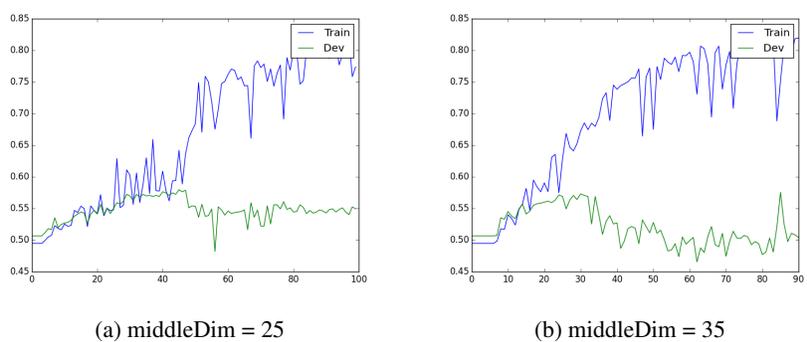


Figure 5: Examples of how middleDim affects performance in RNN 2

From the confusion matrix, we can see that the model is still nor good at predicting the negative labels, it has a tendency to mislabel positive as negative, which is not a surprise as the positive training data has a dominating amount. Despite the average dev accuracy of RNN2 is not as good as RNN, it has improved in the prediction of neutral and positive labels by mislabeling less positive instance as neutral labels.

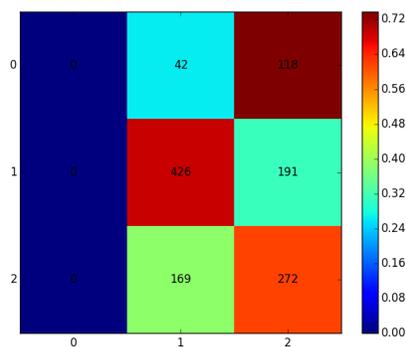


Figure 6: RNN 2 confusion matrix

RNTN:

Theretically speaking, RNTN could have been performing better than RNN and RNN2. However, due to the lack of word- and phrase- level in the dataset, RNTN model is under-fit. With other hyperparameters tuned to its best, we try to adjust the dimension of the middle hidden layer to have the model properly fit the data. We can see in Figure 7 that, apparently, the lower dimension performs better.

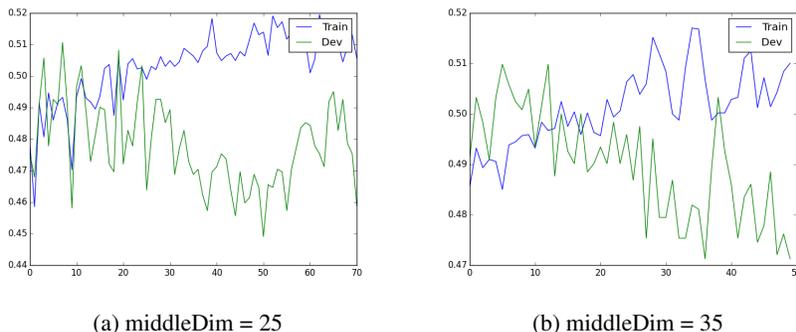


Figure 7: Examples of how middleDim affects performance in RNN 2

Results at a glance:

Models	Dev ACC	Avg F1 scorE
One-hidden-layer RNN	63.71	0.512
Two-hidden-layer RNN	62.45	0.517
RNTN	59.32	0.483

For referenece, when running the same models on tree bank, the accuracy on dev set is as follows. We can see that with better-labeled data set, these models can generate quality performance.

Models	Dev ACC
One-hidden-layer RNN	84.17
Two-hidden-layer RNN	80.68

5 Conclusion

In summary, sentiment analysis in twitter data strikes for cautious pre-processing and the proper model that best fits the data set. Balance of the data set and available labels of intermediate levels play a significant roles in training such models. Imbalance of our data set lead to a poor performance in predicting negative labels through out the models, and the insufficient intermediate-level (word and phrase- level) labels lead to an under-fit in RNTN.

Another take-home lesson would be tuning the hyperparameters for a better data fit. Our data set could overfit shallow neural nets, such as the one-hidden-layer RNN. By increasing the regularization strength, we are able to obtain a decent performance on one-hidden-layer RNN.

Continuous work after this project to perfect the models could be experimenting with more data set to look for a proper fit that contains more intermidiate-level information and more fine-tuning on the hyperparameters, some of which largely depend on the nature of the data set.

6 Reference

- [1] Agarwal, Xie, Vovsha, Rambow, Passonneau. (2011) Sentiment analysis of Twitter data. *Proceedings of the Workshop on Language in Social Media (LSM 2011)*
- [2] Goller, A. Kuchler. (1996) Learning task-dependent distributed representations by backpropagation through structure. *Proceedings of the International Conference on Neural Networks (ICNN-96)*.

- [3] Klein, Manning. (2003) Accurate Unlexicalized Parsing. *Proceedings of the 41st Meeting of the Association for Computational Linguistics* pp. 423-430.
- [4] Socher, Perelygin, Wu, Chuang, Manning, Ng, Potts. (2013) Recursive deep models for semantic compositionality over a sentiment treebank. *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*. Vol. 1631.
- [5] Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15
- [6] Script for preprocessing tweets by Romain Paulus. <http://nlp.stanford.edu/projects/glove/preprocess-twitter.rb> Retrieved on April 30th 2015.
- [7] Semeval 2013 Task 2 Data. <http://www.cs.york.ac.uk/semeval-2013/task2/>. Retrieved on April 28th 2015.
- [8] Twitter GloVe word vectors. <http://nlp.stanford.edu/projects/glove/> Retrieved on May 6th 2015.