Deep Learning for Sentiment Analysis

Matthew Hauser, Yating Jing
mhauser5@jhu.edu, yating@jhu.edu

1 Summary

The primary goal of this project is to explore the potential of word vectors and integrate the Recursive Neural Network into the binary classifer, to predict the sentiment label given a movie review. We also applied Logistic Regression classification to the word vectors as a baseline model, for the purpose of evaluating the performance of the RNN classifier.

2 Introduction

The application of sentiment analysis to text is a popular area of research in natural language processing and computational linguisitcs. The basic task of sentiment analysis is to identify the *polarity* of a piece of text. It seeks to identify the author's opinion, emotion or intent in the text. Sentiment analysis has a vast array of applications with both public data sets and social media.

The task at hand is a simple binary classification, labeling a movie review as either positive or negative. A number of different classifiers could be used for this task, including linear regregression, support vector machines, naive bayes, etc. However, these models cannot capture semantic information and meaningful attributes. To resovle this problem, word vectors and deep learning technqiues should be integrated into classifer.

In [4], several machine learning applications including movie review sentiment analysis are carried out using word vectors. In the IMDB review classification experiment, a linear SVM is employed in different settings. The model utilizing bag of words representation and trained on extra unlabeled data shows superior performance.

In [2], one-hot word vectors are fed into a RNN encoder-decoder for a machine translation decoding task. The word embedding experiment conducted in the paper shows the ability of word vectors and recursive neural networks to capture the semantic similarities among words, where the words of similar semantic meanings tend to cluster together.

3 Logistic Regression (Baseline Model)

3.1 Assumptions

The following assumptions are made in this model:

1. Movie reviews are independent from each other, then the MAP estimation problems for different reviews are also independent.

2. Words in a movie review are conditionally independent given the mixture variable θ.

3.2 MAP Problem Definition

Given a movie review r which contains N words, probability of r can be represented as:

$$p(r) = \int p(r, \theta)\, d\theta = \int p(\theta) \prod_{i=1}^{N} p(w_i \mid \theta) d\theta \tag{1}$$

Here introduce a Gaussian prior on parameter θ.

Concatenate the word vectors described in the previous section into a word representation matrix $R \in \mathbb{R}^{(\beta \cdot |V|)}$ where each word in the vocabulary V has a β-dimensional vector representation $\phi_w = Rw$, which corresponds to the word's column in R. θ is a β-dimensional weight vector which weights each word's representation vectors. The energy function for each word, w is then

$$E(w; \theta, \phi_w, b_w) = -\theta^T \phi_w - b_w \tag{2}$$

where b_w is a bias for each word.

Use softmax function for the word distribution $p(w \mid \theta)$, we can get

$$\begin{aligned} p(w \mid \theta; R, b) &= \frac{\exp(-E(w; \theta, \phi_w, b_w))}{\sum_{w' \in V} \exp(-E(w'; \theta, \phi_{w'}, b_{w'}))} \\ &= \frac{\exp(\theta^T \phi_w + b_w)}{\sum_{w' \in V} \exp(\theta^T \phi_{w'} + b_{w'})} \end{aligned} \tag{3}$$

Thus, using *maximum a posteriori* (MAP) the learning problem becomes

$$\max_{R,\, b} \prod_{r_k \in D} p(\hat{\theta}) \prod_{i=1}^{N_k} p(w_i \mid \hat{\theta}_k; R, b) \tag{4}$$

where D denotes the total training data, $\hat{\theta}$ denotes the MAP estimate of θ for review r_k.

3.3 Sentiment Model

We give each movie review a binary sentiment label s, positive($s = 1$) or negative($s = -1$). With a predictor function $f(x)$, we can map a word vector ϕ_w to a predicted sentiment label $\hat{s}$:

$$\hat{s} = f(\phi_w) \tag{5}$$

Currently we employ a logistic regression as the predictor. Let $\psi \in \mathbb{R}^{\beta}$ denote the regression weight vector,

$$p(s = 1 \mid w; R, \psi) = \sigma(\psi^T \phi_w + b_c) \tag{6}$$

where $\sigma(x)$ denotes the logistic function and b_c is a scalar bias.

Then given the training dataset of reviews D where s_k denotes the sentiment label for review r_k, the log-objective we wish to maximize becomes

$$\max_{R,\, \psi,\, b_c} \sum_{k=1}^{|D|} \sum_{i=1}^{N_k} \log p(s_k \mid w_i; R, \psi, b_c) \tag{7}$$

3.4 Training

The final objective of the model is the sum of two objectives above:

$$\begin{aligned}\mathcal{L} = &\sum_{k=1}^{|D|}\sum_{i=1}^{N_k}\Big(\log p(\hat{\theta}) + \log p(w_i \mid \hat{\theta}_k; R, b)\Big) \\ &+ \sum_{k=1}^{|D|}\sum_{i=1}^{N_k}\log p(s_k \mid w_i; R, \psi, b_c)\end{aligned} \tag{8}$$

Use the training method proposed in [4] for this non-convex problem. First optimize the word representation (R, b, ψ, b_c) while fixing the parameter $\hat{\theta}$. Then searching for the new $\hat{\theta}$ while leaving (R, b, ψ, b_c) fixed, repeat this procedure until convergence.

4 Recurrent Neural Network

4.1 Introduction to RNN

A recurrent neural network(RNN) consists of an input layer $\mathbf{x}$, hidden layer $\mathbf{h}$ and an output layer $\mathbf{y}$, which operates on a variable-length sequence $\mathbf{x} = (x_1, \ldots, x_t)$, where x_t denotes the input variable at time step t. At each time t, the hidden layer $\mathbf{h}^{(t)}$ is updated by

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, x_t) \tag{9}$$

The hidden state h_t of the RNN at time t is able to encode all the history from $1 \to t-1$.

Given an observed sequence X(a sequence of words), each word fully depends on all the preceding words, because the current hidden status h_t could capture all history preceding current time step. So the probability of observing sequence $\mathbf{x}$ can be written as

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_1) \tag{10}$$

For a specific x_t in a given review $\mathbf{r}$, the probability that $\mathbf{r}$ has the j^{th} sentiment label of K labels is then

$$p(x_t, s = j \mid x_{t-1}, \ldots, x_1)$$

where s denotes the sentiment label of $\mathbf{r}$.

The distribution over $\mathbf{r}$ then becomes a multinomial distribution over K possible outputs. Each output label is represented as a word vector using 1-of-K encoding:

$$p(r = 1 \mid x_{t-1}, \ldots, x_1) = \frac{\exp\{w_j h_t\}}{\sum_{k=1}^{K}\exp\{w_k h_t\}} \tag{11}$$

where K is the number of possible sentiment labels, which is 2 in our case.

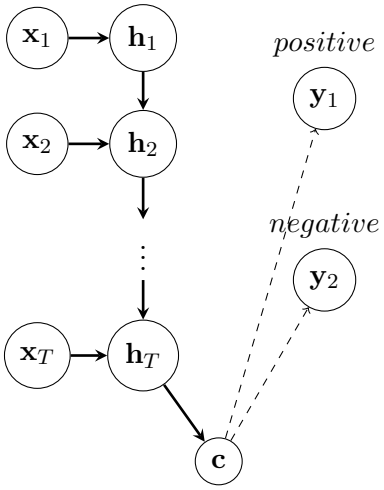

Figure 1: RNN classifier

The RNN takes in a variable length input sequence $\mathbf{x}$, and outputs a summary $\mathbf{c}$. More specifically, given an input sequence $\mathbf{x} = (x_1, \ldots, x_T)$, the update function for the hidden layer $\mathbf{h}^{(t)}$ at time step t is updated using equation (2), where f is the *sigmoid* activation function.

Afterwards, the hidden states of the RNN are summarized by a summary vector $\mathbf{c}$ of the whole input sequence, that is, the whole history of the review is memorized. The summary at the T^{th} step is calculated as below:

$$\mathbf{c} = \tanh(\mathbf{V}\mathbf{h}^{(T)}) \tag{12}$$

where $\mathbf{V}$ is a weight matrix. Note that the summary vector $\mathbf{c}$ is produced when $t = |\mathbf{x}|$.

The conditional probability of the sentiment label s is:

$$p(s \mid \mathbf{c}) = g(\mathbf{h}^{(t)}, \mathbf{c}) \tag{13}$$

where g is the *softmax* activation function.

4.2 Over Structure

The layers of the recursive neural network are illustrated in Figure 2. At each time step, each word of a given review $\mathbf{r}$ would be passed through the input layer and then the hidden layer. This process will recursively loops between the input layer and the hidden layer until all the words in the review are processed. Note that along with the word vector from the input layer, information from previous hidden layers will also be passed on to the current hidden layer.

At time step $t = |\mathbf{r}|$, the information of the entire input review is then passed as a summary vector to the output layer, which will then be mapped into a two dimensional space.

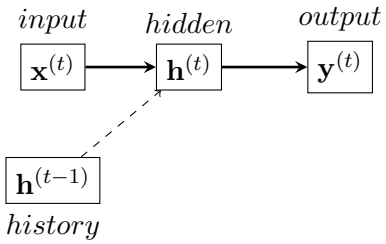

Figure 2: Layer illustration

The activation functions of the three layers are listed below:

a. Input layer: one-hot vector encoder

b. Hidden layer: sigmoid function

c. Output layer: softmax function

We then pick the sentiment label for the review according to the confidence values associated with the two different elements in the output vector, which is calculated uisng *softmax* function.

4.3 Training

4.3.1 Back Propagation

Neural networks can be trained by stochastic gradient descent using back-propagation(BP) algorithm as proposed in [5]. Assume the weights for input, hidden and output layers are represented by matrices **U**, **W** and **V**. The weight matrices are adjusted after seeing every example. A cross entropy criterion is used to obtain gradient of an error vector in the output layer, which is then back-propagated to the hidden layer, then to the input layer. The illustration of one epoch of a simple RNN is shown in Figure 3.

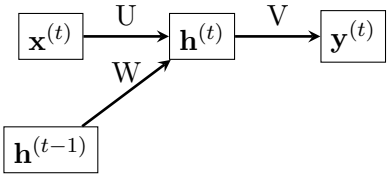

Figure 3: Illustration of one epoch of RNN

The RNN classifier is trained to maximize the following conditional log-likelihood:

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log\, p_\theta(\mathbf{y}_n = \mathbf{d}_n \mid \mathbf{R}_n) \tag{14}$$

where $\mathbf{R} = (R_1, \ldots, R_N)$ is the movie reviews, $\mathbf{y} = (y_1, \ldots, y_N)$ denotes the sentiment output vector, and $\mathbf{d} = (d_1, \ldots, d_N)$ is the correct label vector.

The goal above is then equivalent to minimize the difference between the prediction and the true label at each time step. Here define the loss(error term) at each time step as the *xor* operation of the prediction and the label:

$$\mathcal{L}^{(t)} = \mathbf{d}^{(t)} \oplus \mathbf{y}^{(t)} \tag{15}$$

The gradient of error vector in the output layer $\mathbf{e}_o^{(t)}$ w.r.t a given weight matrix $\mathbf{m}$ at each time step is then:

$$\mathbf{e}_o^{(t)} = \frac{d}{d\,\mathbf{m}} \mathcal{L}^{(t)} \tag{16}$$

where $\mathbf{m}$ denotes one of the weight matrices **U**, **W**, **V**.

Therefore each element of the weight matrix **V** is adjusted as follows:

$$v_{jk}^{(t)} = v_{jk}^{(t-1)} + \eta\, h_j^{(t)} e_{ok}^{(t)} \tag{17}$$

where η is the learning rate and $e_{ok}^{(t)}$ is the error gradient of the k^{th} neuron in the output layer.

Then back propagate the error vector from the output layer to the hidden layer using the equation below:

$$\mathbf{e}_{hj}^{(t)} = \mathbf{e}_o^{(t)^T} \mathbf{V}\, h_j^{(t)}(1 - h_j^{(t)}) \tag{18}$$

Next update the weight matrix for the input layer $\mathbf{U}$:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} + \eta\, x_i^{(t)} e_{hj}^{(t)} \tag{19}$$

The update function for the recurrent weight matrix $\mathbf{W}$ is:

$$w_{lj}^{(t+1)} = w_{lj}^{(t)} + \eta\, h_l^{(t-1)} e_{hj}^{(t)} \tag{20}$$

4.3.2 Back Propagation Through time

For recurrent neural network training, the BP algorithm is not optimal, in that the network only tries to optimiaze the prediction of the next word given the previous word and previous state of the hidden layer. To extend such effect into the future, the short term information stored in the hidden layer can be replaced by some long context information.

According to [5], the back-propagation through time(BPTT) training algorithm can ensure that the network will learn what information to be stored in the hidden layer. A RNN with one hidden layer which is used for N time steps can be seen as a deep feedforward network with N hidden layers, which is illustrated in Figure 4, where the unfolding is applied for three time steps.

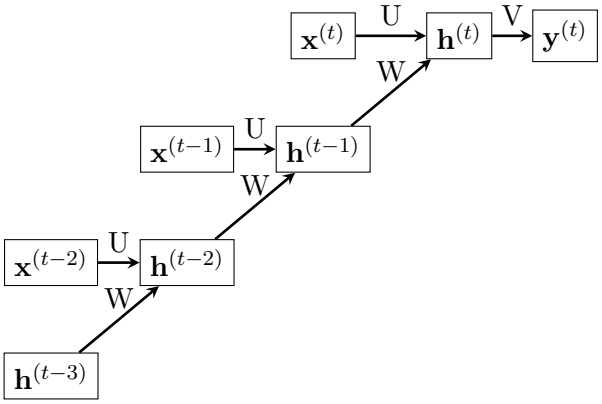

Figure 4: Illustration of one epoch of RNN as a deep feedforward network

In such RNN errors are propagated from the hidden layer $\mathbf{h}^{(t)}$ to the hidden layer from the previous time step recurrently as follows:

$$\mathbf{e}_h^{(t-\tau-1)} = d_h(\mathbf{e}_h^{(t-\tau)^T} \mathbf{W}, t - \tau - 1) \tag{21}$$

where the error vector is obtained using element-wise function $d_h()$:

$$d_{hj}(x, t) = x h_j^{(t)}(1 - h_j^{(t)}) \tag{22}$$

This unfolding process can be applied for as many time steps as many training samples were already seen, however a certain number of steps of unfolding would be sufficient since error gradients quickly vanish as back-propagated in time.

The weight matrix $\mathbf{U}$ between the input layer and hidden layer is then updated as:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} + \sum_{\tau=0}^{T} \eta \, x_i^{(t-\tau)} e_{hj}^{(t-\tau)} \tag{23}$$

where η is the learning rate. Note that $\mathbf{U}$ must be updated in one large change instead of during an incremental changing process, which can lead to instability.

The update function for the recurrent weight matrix $\mathbf{W}$ is then:

$$w_{lj}^{(t+1)} = w_{lj}^{(t)} + \sum_{\tau=0}^{T} \eta \, h_l^{(t-\tau-1)} e_{hj}^{(t-\tau)} \tag{24}$$

5 Experiment

5.1 Dataset

The data is provided by the Kaggle competition: *Bag of words meets bags of popcorn*, which consists of reviews from the Internet Movie Database, IMDB. The reviews are labeled as positive or negative, based on their rating out of ten stars. We have 20,000 labeled reviews for training and 5,000 for evaluation. We removed some punctuations from the movie reviews.

5.2 Implementation

We utilized *Theano* API [1] to implement the RNN model. The initial weights of all three weight matrices are sampled from normal distribution $\mathcal{N}(0, 0.01)$. The number of hidden units in the hidden layer is set to be 60. The learning rate in the *Stochastic Gradient Descent* step is 0.01.

5.3 Results and Evaluation

For the logistic regression model, we obtained a baseline result of 0.79, which is very satisfying. However, due to the substantial time needed for the training process, we are only able to run our RNN model on a small portion of the entire dataset. With only 160 training reviews and 40 test reviews, we can get an accuracy of 0.62 while the logistic regression model gives an accuracy of 0.59.

Even with only 40 training reviews and 10 test reviews, RNN classifier predicts 60% of the test reviews correctly, which shows its strong learning ability. This is also because that each review contains several hundreds even more than a thousand words, the model would then be exposed to a large vocabulary and many possible contexts even with a limited number of training reviews. On the other hand, the large vocabulary makes the size of the input matrix of each review very huge, which could lead to memory blowup when trained on low performance computers.

To fix the problem above, we could consider removing some irrelavent words such as *I*, *you*, *and*, etc, to make the reviews more concise. Also, we could limit our vocabulary so as to reduce the size of each word vector. Additionally, we could consider add a **word embedding** layer to the network and map the input vectors to a much smaller vector space in order to boost the RNN performance.

5.4 Conclusion and Forthcoming Research

Although only trained on a small dataset, the RNN model still demonstrates its capabilities. Encoding the words into vectors, the logistic regression model also gives very satisfying results.

The future work would be to better preprocess the training data and upgrade the model in order to increase its efficiency.

To further improve the classification accuracy, rather than just feeding the one-hot word vectors into the RNN, we can ustilize Google **Word2Vec** tool, [3], which provides an efficient implementation of the continuous bag-of-words and skip-gram architectures for computing word representations. Current studies show that such word vectors capture many linguistic regularities. *Word2Vec* contains a tool which can provide the *distance* to another word based upon training data. In the example of the word *France* the output contains:

word	distance
spain	0.678515
belgium	0.665923
netherlands	0.652428
italy	0.633130
⋮	⋮
mango	0

Moreover, in addition to the RNN model architecture described above, we can adopt a simplified version of **long-short term memory**(LSTM) units as the hidden units, which allows the hidden layer to drop any irrelevant information and control how much information from previous hidden states to be passed on. In this way we will be able to ignore irrelevant words such as *I*, *you*, *and*, etc. According to [2], such hidden units are crucial in getting meaningful results in the task of translation decoding.

References

[1] Frederic Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua. Bengio. Theano: new features and speed improvements. *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.*, 2012.

[2] van Merrienboer B. Gulcehre C. Bougares F. Schwenk H. Bengio Y. Cho, K. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 2014.

[3] Google. word2vec. https://code.google.com/p/word2vec/.

[4] Andrew L. Maas. Learning word vectors for sentiment analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 1, 2011.

[5] T. Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View*, April 2012.