COMPGI19 Assignment3 – Sentence Representations for Sentiment Analysis on Twitter

Weijie Huang (15042248)
weijie.huang@ucl.ac.uk
Shuo Wang (15033967)
ucabssw@ucl.ac.uk
Dongdong Zhang (15021014)
dongdong.zhang.15@ucl.ac.uk
MSc Web Science and Big Data Analytics

Abstract

In this assignment, students are required to study the representation learning of sentiment analysis of Tweets. After that, a simple deep learning structure is built. This report includes four problems: the gradient checking, sun-of-word-vectors model, recurrent neural networks and free optimized solution. In addition, the basic element which can be used to achieve the main learning method backpropagation is "Block", and the "Block" has three basic sections: forward, backward and update.

1 Problem 1: Gradient Checking

To make sure the correctness of gradients calculated by blocks, GradientChecker is used to get the gap between gradients by blocks and the actual gradient value which is obtained by the following equation:

$$\frac{\partial g_\theta}{\partial x_i} \approx \frac{g_\theta(x+i\epsilon)+g_\theta(x-i\epsilon)}{2*\epsilon}$$

In the program, if the difference between them is too large over 10^{-6}, it will present the gradient checking failed. On the other hand, it will show the average error, which means that the implemented blocks are correct and can be used in the further development. Although, using the above equation can calculate the gradient, but it is not a good idea. It is the reason that the loss function is called twice to obtain the gradient value, but generally, it only needs calling the loss function once. If the data size and iteration time are large, the calculation time could be long. Besides, it is not accurate enough, just equals to the real result approximately no more than 10^{-6}.

The blocks are tested by this functions with simple data to verify the validity. The average error of Dot block is $2.344*10^{-10}$, $2.315*10^{-10}$for Sigmoid block, $2.962 * 10^{-10}$ for NegativeLogLikelihoodLoss block and $1.758 * 10^{-11}$ for L2Regularization block. Therefore, the developed blocks are correct.

2 Problem 2: Sum-of-Word-Vectors Model

2.1 Blocks

There are six blocks in Sum-of-Word-Vectors Model, where the first block Vector Parameters provides a trainable parameter and gradient clipping, and the following three blocks calculate the score of the sentence, shown as followed. And the last two blocks are loss function and regularization function which are used to get the final value of loss and support Stochastic Gradient Descent to find the best parameter. The below function is used to show the general method of this problem.

$$f_\theta(x_{1,} \dots, x_N) = sigmoid(\boldsymbol{w} * \sum_i \boldsymbol{v}_{\boldsymbol{x}_i})$$

2.1.1 Vector Parameters

This block is the basic one, used for $\boldsymbol{w}$ and $\boldsymbol{v}_{\boldsymbol{x}_i}$. Firstly, it initializes two vectors *param* and *gradParam*, where the first records the current parameter, and the latter records the gradient which will be used to update *param*. There are five methods in this class. *forward()* caches the current value to output; *backward()* accumulates gradient in *gradParam*. In *update() function*, the gradient will be limited to the range (-clip, clip), and then it will be used with *learningRate* to update *param*.

2.1.2 Vector Sum

This block represents the sum of word vectors to get a sentence vector, where *forward()* calculates the summation of vectors, and *backward()* calculates the derivative of the summation and the upstream gradient. Because the derivative of summation is a scalar one, so it just calls the upstream gradient. Besides, call the update method in *update()* with input value *learningRate*.(Same process in *update() in following blocks*)

2.1.3 Dot Product

Similar to Vector Sum block, where *forward()* calculate the dot product of two vectors, and *backward()* calculates the derivative of dot product and the upstream gradient z which includes the derivation of x and y individually, $z\frac{\partial x*y}{\partial x} = z * y$ and for $z\frac{\partial x*y}{\partial y} = z * x$.

2.1.4 Sigmoid

The Sigmoid function that is shown in equation (1), is the last step to get the score of the sentence, and the output of the last step is regarded as the input in the Sigmoid function to produce an output between 0 and 1. This score is the classification of a Twitter sentence.

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{1}$$

Therefore, in the *forward()* method, it uses the equation (1) to scalar the value. At the same time, the *backward()* method are relative to use equation (2) to calculate the deviation of the Sigmoid function. Furthermore, it reports the multiplicative value of upstream gradient and deviation, $z * \frac{\partial sigmoid(x)}{\partial x} = z * Sigmoid(x) * (1 - Sigmoid(x))$.

$$Sigmoid'(x) = \frac{1}{1+e^{-x}} * \frac{e^{-x}}{1+e^{-x}} = Sigmoid(x) * (1 - Sigmoid(x)) \quad (2)$$

2.1.5 Negative Log-likelihood

After getting the score of the sentence calculated by these three blocks above, a loss function should be built with the target values, which is called negative log-likelihood to find the difference between the prediction and target classification. The *forward()* method applies the equation (3) to calculate the loss value.

$$\mathcal{L}(f_\theta(x), y) = -y * \log\big(f_\theta(x)\big) - (1 - y) * \log\,(1 - f_\theta(x)) \quad (3)$$

In addition, the *backward()* method uses equation(4) to calculate $\frac{\partial \mathcal{L}(f(x),y)}{\partial f(x)}$. It is different with above questions that it does not need to multiply the upstream gradient, because the loss function is the root node.

$$\frac{\partial \mathcal{L}(f(x),y)}{\partial f(x)} = -\frac{y}{f(x)} + \frac{1-y}{1-f(x)} \quad (4)$$

2.1.6 l_2 Regularization

To avoid overfitting and underfitting, l_2 regularization is added with the loss function, and showed as on equation (5). In *forward()* method, for each element in args, it transfers the matrix to vector by toDenseVector if needed. After that, it calculates each element in $||\theta||^2$, and sum it up at the end and get $\lambda * \frac{1}{2}||\theta||^2$.

$$\mathcal{R}(\theta) = \lambda * \frac{1}{2}||\theta||^2 \quad (5)$$

In *backward()* method, use equation(6) to calculate $\frac{\partial \lambda * \frac{1}{2}||\theta||^2}{\partial \theta}$, which is presented in the equation (6).

$$\mathcal{R}'(\theta) = \lambda * ||\theta|| \quad (6)$$

2.2 Model

In the Model section, the individual word is represented as a vector which can be calculated in the further steps, and then a sentence vector is formed by several word vectors. Moreover, it calls the Sigmoid and L2Regularization function completed in the Block section to obtain the score of this sentence and loss after adding regularization. Therefore, in the main function, the SumOfWordVectorsModel is called to return the final loss of the whole Twitter sentences. Besides, in Problem 1, it checks the validation of each block, and shows the blocks are correct for the gradient. Therefore, in this problem, the whole model is tested with the gradient check. To achieve that, an example sentence is read with the tag, “On my way home from Orlando – such a great time had” with 1 tag for positive. The final average error of the test section is about 6.574×10^{-10}. Therefore, the model could be correct for applying the massive data set.

2.3 Stochastic Gradient Descent

The section is used to apply the learning method with the loaded data. It has an accumulated loss variable, and there is nested loop to add the loss to the variable. Also, the sentence and the responding target are loaded, and then call backword/update to update the weight parameter. This function is called at the main function with five inputs variables, including the model, the training set name, the iterator time, and epochHook; then it returns a new epochHook to display results.

2.4 Grid Search and Evaluation

To find the best results, the various hyperparameters are adjusted to test, including the learning rate and regularization strength. In fact, the initial learning rate is 0.01, and the regularization strength is 0.01. With these parameters, although on the front half part, it makes the learning normally, the accumulated loss reduced and train/dev accuracy increased. Another half part obtains NaN results of loss, and the accuracy becomes lower rapidly. In this case, the training accuracy is around 100%, but the dev does not have large improvement. Therefore, these phenomena show that it has overfitting on training the model.

To solve this problem, the learning rate are changed to 0.009 and the regularization strength becomes 0.02. It reduces the learning ratio and increases the strength of the regularization to avoid the overfitting. The small learning rate makes each update on small step to avoid missing the best value. In addition, the large regularization strength makes the model reducing the overfitting. The accumulated loss is around 18119.951; the accuracy of training set is about 99.45; and the accuracy is 75.83 approximately. It removes the NaN of the loss accumulated, but the train accuracy is still too closed to 100%. So it improves the results further to obtain a better dev accuracy. Following the increment of the regularization strength, the accuracy of train is decreased, but the dev accuracy is increased. In this problem, the best configuration is learning rate about 0.002 and regularization about 0.08. The loss value is 55149.680 and accuracy of train/dev becomes 90.58/77.36. It obtains the maximum dev accuracy value.

Learning Rate	Regularization Strength	Loss accumulated	Train accuracy	Dev accuracy
0.01	0.01	NaN	41.62	40.28
0.009	0.02	18119.951	99.45	75.83
0.005	0.05	34964.418	97.89	76.23

0.002	0.08	55149.680	90.58	77.36

2.5 Loss Analysis

In this section, it plots the figure with the best configuration. The x-axis of the both graphs are epoch time, but for y-axis one is the accumulated loss, and one is accuracy. Figure 1 shows that following the epoch time increased the accumulated loss is reduced which has improvement rapidly and then it reduced as a line slowly. It means that the model is improved after each update, and the value might achieve the best value after more epoch time.

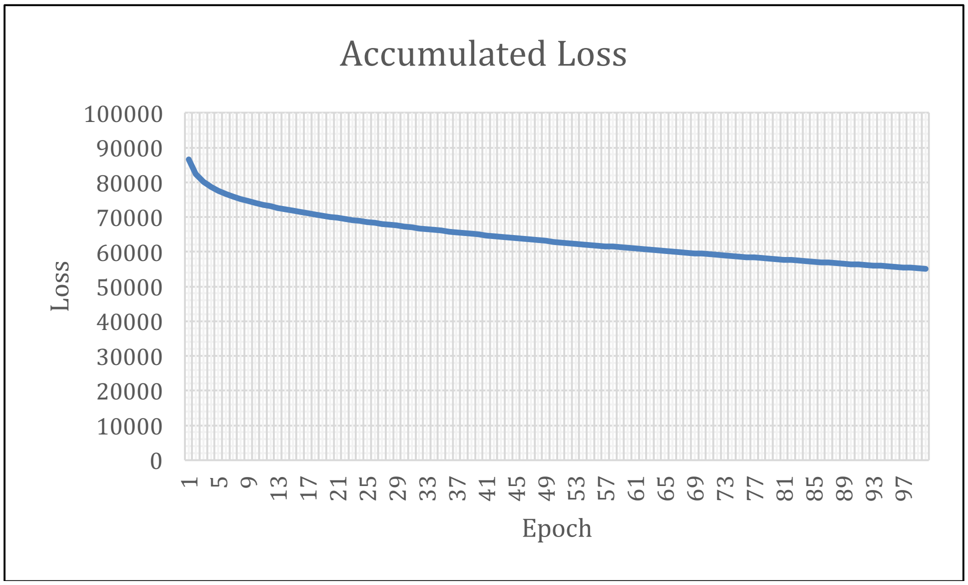


Figure 1: Epoch VS Accumulated loss

In Figure 2, it presents the accuracy of train and dev data. The accuracy is increased with epoch time, increasing fast at first and then following a stable line. It is similar to the trend of Figure 1, but it has a positive slope. Moreover, the accuracy of the training set is always larger than dev dataset, because the training data is used to train the model which is applied to find the predication accuracy of the train and dev data. The results of Figure 1 represent that there is no NaN on the whole training processing, which means that the model is not in overfitting. Also, the accuracies of the train and dev are improved to a high level; so it is not in underfitting.

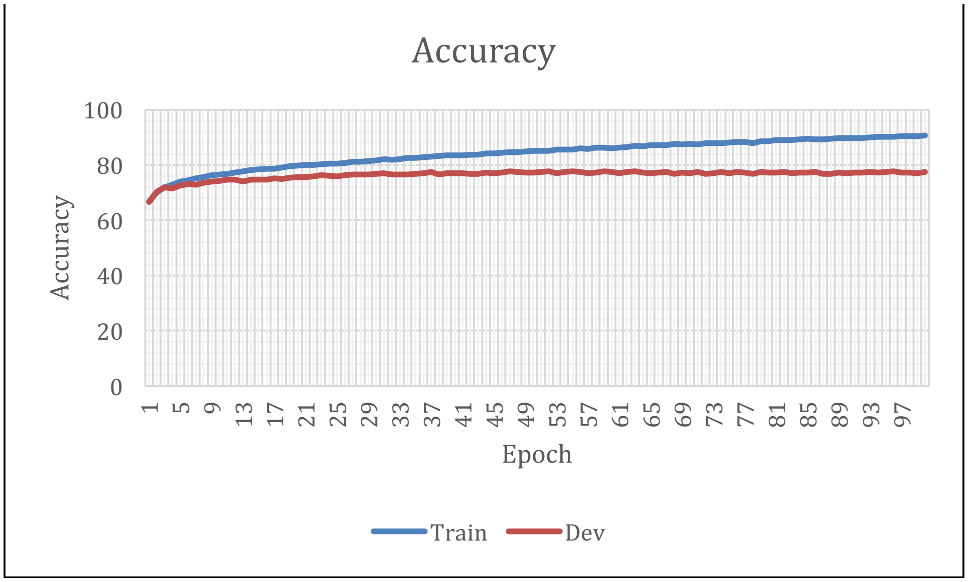


Figure 2: Epoch VS Accuracy

2.6 t-SNE Visualization

The library chosen is implemented in Java. To visualize the trained word representations, the first step is outputting two vectors, words vector and the corresponding weight vector, to two txt files. Secondly, call these files in Java, and tune the format of visualization to color the words based their weight.

Unfortunately, after outputting these two files, the code can't match our data successfully.

3 Problem 3: Recurrent Neural Networks

3.1 Blocks

There are three blocks added in this part to get the vector of the sentence. Unlike the Sum-of-Word-Vectors Model, RNN model takes the word's order into account. The method is shown below:

$$\boldsymbol{h}_t = tanh\,(\boldsymbol{W}^h \boldsymbol{h}_{t-1} + \boldsymbol{W}^x \boldsymbol{x}_t + \boldsymbol{b}) \tag{7}$$

In this equation, $\boldsymbol{W}^h$ represents the recurrent weight matrix, $\boldsymbol{h}_{t-1}$ is the history in step t-1, $\boldsymbol{W}^x$ and $\boldsymbol{x}_t$ are inputs, and $\boldsymbol{b}$ is bias. Thus, the number of iteration and hidden layer is the number of words in a sentence, and the last hidden vector $\boldsymbol{h}_N$ is the vector of sentence.

3.1.1 Matrix Parameter

Similar with Vector Parameter in Problem 2, but for matrix. At first, it initializes two matrixes *param* and *gradParam*. Other processes are same with *VectorParameter* block.

3.1.2 Matrix-Vector Multiplication

Similar with the *Dot* block mentioned above. In *forward()* method, the output is a matrix multiplying a vector, which involves the matrix multiplication rather than dot. It produces a vector of two inputs. In *backward()* method, $z\frac{\partial Wx}{\partial x} = z * W$ and $z\frac{\partial Wx}{\partial W} = z * x$. Outer product should be used to calculate $z * x$, because z and x both are vectors, which will obtain a matrix.

3.1.3 Tanh

Tanh is a function making every element in a vector to the range (-1,1). In *forward()* method, a addition vector with three intermediates interacted is inputted, and then get squashed into a new state vector using *tanh* function provided. In *backward()* method, as represented in equation 8, $z\frac{\partial \tanh(x)}{\partial x} = z(1-(\tanh(x))^2)$.

$$\frac{\partial \tanh(x)}{\partial x} = 1-(\tanh(x))^2 \tag{8}$$

3.2 Model

In Recurrent Neural Networks model, the way to get sentence vector is changed, precisely, taking word's orders into account using history. In equation(7), $\boldsymbol{h}_t$ is updated with two inputs $\boldsymbol{h}_{t-1}$ and $\boldsymbol{X}_t$, where $\boldsymbol{h}_{t-1}$ is history. Finally, $\boldsymbol{h}_N$ represents sentence vector with dimension hiddenSize. Then, the score of the sentence is calculated with the same way in Problem3, and the final summation with regularization score by L2Rularization function is obtained.

Besides, in Problem1, the whole model is tested. The same example is used, "On my way home from Orlando – such a great time had" with 1 tag for positive. The final average error is about 3.322×10^{-10}. Therefore, this model is available.

3.3 Grid Search, Initialization, Evaluation and Comparison

To find the best configuration, the regularization strength parameters of both vector and matrix are set to 0.01 which is fixed, and also the word and hidden dimension is 10 for all the test examples. The learning rate is reduced from the initial value 0.01. Otherwise, initializations of parameter vectors/matrices remain GaussianDefaultInitialization, with mean 0.0 and standard deviation 0.3 from a random number. Comparing with other initialization methods, like a random number between 0 and 1, GaussianDefaultInitialization has a better performance. In the table, the best learning rate is 0.0017 used in the further.

Learning Rate	Regularization Strength	Loss accumulated	Initializations	Train accuracy	Dev accuracy
0.01	0.01	NaN(44)	Gaussian*0.2	41.62	40.28
0.0017	0.01	67764.1135	Gaussian*0.2	78.02	73.83
0.0020	0.01	NaN(93)	Gaussian*0.2	43.66	40.98

Comparing with Sum-of-Words, the result is even slightly less. It is because the dataset is not big enough, thus, RNN can't perform very well under this situation.

4 Problem 4: Sky's the Limit

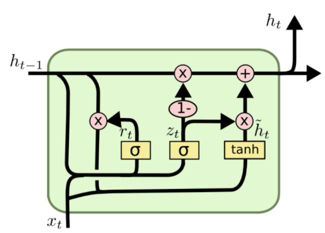
In this part, pre-trained word2vec and LSTM model are used.

Firstly, pre-trained word2vec is an efficient implementation of continuous bag-of-words and skip-gram architectures for vector representations of words. In problem 2&3, the results are always constant after a long time running using the Gaussian default word vector, thus, we tried to apply this method to initialize word vectors to cut the running time to get a good performance.

Secondly, to get sentence vector, Gated Recurrent Unit (GRU) is used, introduced by (Cho, et al., 2014) [5], a slightly more dramatic variation on LSTM. LSTM is a special kind of RNN, designed to avoid the long-term dependency problem, and remembering information for long periods of time. Moreover, GRU is simpler than the standard LSTM models, because it combines the forget and input gates into a single "update gate", and merges the cell state and hidden state. Thus, it is easier to implement.

Finally, we changed the dimension to 25, according to a research paper. (Lai, S, et al, 2015) [7]

Instead of having a single neural network layer, there are four, interacting in the way shown as follows.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 3: Schematic GRU (Christopher, O., 2015) [1]

4.1 Blocks

Apart form blocks in Sum-of-Words and RNN, other three blocks are needed to form a layer.

4.1.1 SigmoidVector

Like Sigmoid block which output a new double within (0.0,1.0) from a double input, Sigmoidvector output a vector with each element within (0.0,1.0) from a vector input.

In *forward()* method, repeat equation(1) and x is a vector in this time. In *backward()* method, use equation(2) to calculate $z * \frac{\partial sigmoid(x)}{\partial x}$ equaling to $z * (Sigmoid(x)\ dot\ (ParamOne - Sigmoid(x)))$, where $ParamOne$ is a vector where each element equals to 1.0, same dimension with vector x.

$$Sigmoid'(x) = Sigmoid(x)\ dot\ (ParamOne - Sigmoid(x)) \tag{9}$$

4.1.2 Minus

In contrast to Sum Block, this represents the subtraction of sequence, where *forward()* calculate the subtraction of vectors, and *backward()*pass the derivative of summation and upstream gradient, $-1 * \boldsymbol{z}$.

4.1.3 Hadamard

Hadamard product is a product of two vectors to a new vector.

In *forward()* method, the product is $x{:}*\,y$. . In *backward()* method, two derivatives are $\boldsymbol{z} * \frac{\partial \boldsymbol{x} \odot \boldsymbol{y}}{\partial \boldsymbol{x}} = \boldsymbol{z} \odot \boldsymbol{y}$ and $\boldsymbol{z} * \frac{\partial \boldsymbol{x} \odot \boldsymbol{y}}{\partial \boldsymbol{y}} == \boldsymbol{z} \odot \boldsymbol{x}.$

4.2 Model

In this model, word vectors are pre-trained, and long-term memory is remembered to get sentence vector. As shown in picture 3, finally, $\boldsymbol{h}_N$ represents sentence vector with dimension hiddenSize. After that, the score of the sentence and the final summation with regularization is calculated in the same way in Problem3. Because of long running time of this model, we are failed to get the final result.

4.3 Model Choice

To analyze three models used in this assignment, methods to get word vector and sentence vector, and the running time is compared shown as follows.

	Sum-of-Words	RNN	Pre-trained +GRU
WordToVector	Guassian Initialization with mean 0.0 and standard deviation 0.2	Guassian Initialization with mean 0.0 and standard deviation 0.2	Word2Vec, representing word's relation to other words
WordVectorToSentenceVector	Simply summation, ignoring word order	Take word order into account by dynamic temporal behavior	Take word order into account and combine long term and short term memory.
Running Time(100 epoches, approximately)	9 minutes	3 hours	a few decades (Very long)

As we can see from this table, from the sum of words to RNN and GRU, it is more complicated and should be more accurate, but the longer time needed.

Two tables 1&2 proof the suppose, accuracy in RNN is higher than Sum-of-Words.

To achieve the prediction result, the model we used to predict is RNN, instead of GRU because of long running time.

References

[1] Christopher, O., 2015. Understanding LSTM Networks. colah.github.io Colah's blog, [blog] 27 Aug. Available at: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> [Accessed 31 Jan 2016].

[2] Mitchell, J. and Lapata, M., 2008, June. Vector-based Models of Semantic Composition. In ACL (pp. 236-244).

[3] Lipton, Z.C., 2015. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019.

[4] Andrej, K., 2015. The Unreasonable Effectiveness of Recurrent Neural Networks. karpathy.github.io Andrej Karpathy blog, [blog] 21 May. Available at: < http://karpathy.github.io/2015/05/21/rnn-effectiveness/> [Accessed 31 Jan 2016].

[5] Chung J, Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv preprint arXiv:1412.3555, 2014.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research (JMLR), 12:2493-2537, 2011.

[7] Lai, S., Liu, K., Xu, L. and Zhao, J., 2015. How to Generate a Good Word Embedding?. arXiv preprint arXiv:1507.05523.