# Toxic Comment Classification

Zhaomin Xiao
School of Computing and Information
Pittsburgh, Pennsylvania
zhx36@pitt.com

Jingying Tang
School of Computing and Information
Pittsburgh, Pennsylvania
jit33@pitt.com

## ABSTRACT

More and more comments are published in every corner of the internet, some of them are suggestive, some of them are useless. Because of the existence of huge amount of toxic comments, we need to classify them to deal with it more efficiently. Thanks for recent huge progress of deep learning, we can use bidirectional LSTM to finish this task. We also use dropout to avoid overfitting and test the score of different optimizers in Kaggle.

## KEYWORDS

Deep Learning, Word Embedding, Recurrent Neural Network, Long Term Dependency, Long Short-term Memory, Optimizer, Dropout

Zhaomin Xiao and Jingying Tang. 2018. Toxic Comment Classification. In *Proceedings of Dec 2018 (Toxic Comment Classification),* Zhaomin Xiao and Jingying Tang . , 5 pages.

## 1 INTRODUCTION

In 2013, Mikolov et al.[15]introduced a new method to predict word in context, which is an advanced method compared with the global matrix factorization methods introduced by Deerwester et al.[7]. The idea behind these models are from the work of Bengio et al [4]. From the perspective of the association between the word and the context, Church and Hanks [6] introduced pointwise mutual information (PMI) which was subsequently replaced by PPMI because for the rare words, the PMI value of it will go infinity. Besides that, Turney and Pantel [22]indicated that the bias of PMI and PPMI towards rare events. Using some kinds of metric like PMI, Levy et al. [14] and Tobias et al. [19]found that the relative advantage stem from word embedding is from the certain system design choices and hyperparameter optimization, not the algorithm themselves. The accuracy of current methods is not good, the reason for that can be divided into several parts. The first one is that the definition of insult, threat, and other kinds of toxicity are not very clear. Labeling data manually will lead to various degree of bias. And the high cost of labeling will limit the amount of accessible data. Secondly, current classification methods are also not perfect. Some researchers use traditional machine learning classification methods, some researchers use deep learning techniques. The choice of algorithms and parameter tuning are still difficult. And some of theories behind deep learning technique are still strange to researchers. Thus, both of these two methods have space to develop.

## 2 RELATED WORK

*Variants of Naive Bayes and Support Vector Machines.* NB and SVM are often used as baseline methods for text classification. In this test, many researchers try to use NB and SVM to classify the toxic comment. But their performance varies greatly depending on the model variant, features used and dataset. We try to use more clear and accurate model to do this test.

*CBOW and Skip-gram models.* The commonality between these two models is that they both use the information from the context to make prediction. The difference between them is that the Continuous Bag-of-Word (CBOW) predict the central word given the previous words and future words, while Skip-gram uses central word to predict the surrounding words. Besides the Mikolov et al., Rong [18] also gave a good explanation about these models. To reduce its complexity, Mikolov et al. [15] presented negative sampling and hierarchical softmax.

*Keras.* Several machine learning frameworks, such as TensorFlow [3], Keras [5]and Caffe [11] have become popular since Geoffrey Hinton, the Godfather of deep learning, ignited the progress in deep learning research. TensorFlow was released by Google in 2015, and Keras was developed by FranÃğois Chollet. Basically, Keras uses TensorFlow as the back-end, uses itself as the front-end, which can reduce the amount of codes and let developers focus on the algorithms themselves.

*Long Short-term Memory.* Long short-term memory was first introduced by Hochreiter and Schmidhuber [10]. The key property of LSTM is its various kinds of gates which can collect and control the information from the current hidden and previous state. Input gate uses the input word and the past hidden state to determine if the input is worth preserving and thus is used to gate the new memory. After ensuring that the input is worth preserving, the new memory is generated by using the input word and the past hidden state. Forget gate also uses the input word and the past hidden state, but to determine if the past memory is useful for the computation of the current memory cell. The final memory is generated by taking the forget gate into account, using the new memory generated before. The output/exposure gate determines what parts of the final memory should be exposed in the hidden state. The detail of LSTM is presented in the Figure 1.

The mathematical formulation behind the LSTM is as follows.

Input gate:

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \tag{1}$$

Forget gate:

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \tag{2}$$

Output/Exposure gate:

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \tag{3}$$

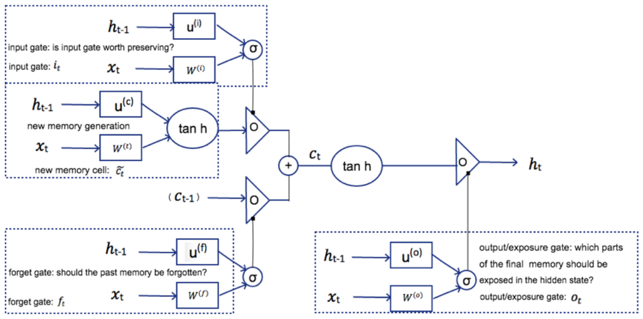

$h_{t-1}$
$u^{(i)}$
input gate: is input gate worth preserving?
input gate: $i_t$
$x_t$
$W^{(i)}$
σ
$h_{t-1}$
$u^{(c)}$
new memory generation
$x_t$
$W^{(t)}$
new memory cell: $\tilde{c}_t$
tan h
$c_t$
tan h
$h_t$
$(c_{t-1})$
$h_{t-1}$
$u^{(f)}$
forget gate: should the past memory be forgotten?
forget gate: $f_t$
$x_t$
$W^{(f)}$
$h_{t-1}$
$u^{(o)}$
$x_t$
$W^{(o)}$
output/exposure gate: which parts of the final memory should be exposed in the hidden state?
output/exposure gate: $o_t$

**Figure 1: LSTM**

New memory generation:

$$c_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) \tag{4}$$

Final memory generation:

$$c_t = (f_t c_{t-1} + i_t c_t) \tag{5}$$

Current hidden state:

$$h_t = o_t + \tan h(c_t) \tag{6}$$

Compared to RNN, LSTM can solve the long-term dependency problem and it is shown in Figure 2. Suppose we try to predict the last words of *I grew up in China... I speak fluent mandarin.* The current information suggests that the next word might be the name of a language, but if we need to figure out what language we are, we need the context of China that is far from the current location. This means that the interval between the relevant information and the current predicted position will certainly become quite large. Unfortunately, as this interval grows, the RNN loses the ability to learn to connect so far.

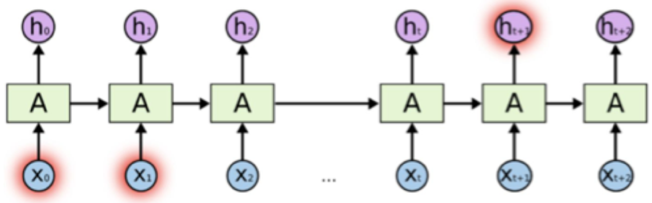


**Figure 2: Long-term dependency**

*Bidirectional LSTM..* The bidirectional LSTM was introduced by Schmidhuber [2]. Bidirectional LSTM has higher efficiency than LSTM. Besides that, more accurate results can also be obtained by using Bidirectional LSTM [20]. It's shown in Figure 3.

*Dropout.* Dropout, introduced by Srivastava et al. [17], is a technology to prevent overfitting. The idea of dropout is very simple. Firstly, the neural network should be sampled within the full neural network. Secondly, only the parameters of sampled neural network need to be updated based on the input data. While training, dropout is implemented by keeping a neural active with the probability p. The detail of dropout is presented in the Figure 4 and Figure 5.

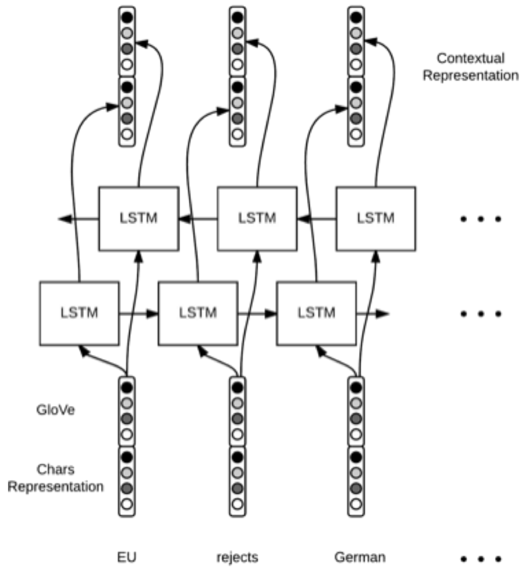


**Figure 3: Bidirectional LSTM Mechanism**

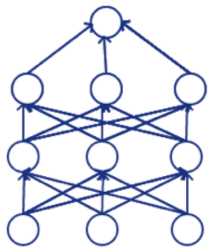


**Figure 4: Standard Neural Network**

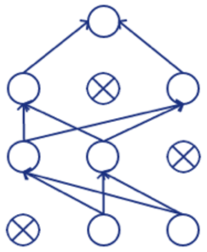


**Figure 5: After applying dropout**

*Gradient descent optimization algorithms.* There are many optimization algorithms for various gradient descent variants. The most infeasible gradient descent is the batch gradient descent. Given the learning rate $\eta$, it computes the gradient of the cost function with respect to the parameter $\eta$ for the entire dataset:

$$\theta = \theta - \eta\nabla_\theta J(\theta) \tag{7}$$

Obviously, it is very inefficient because it computer the gradients for the entire dataset to perform every update. The improved version of batch gradient descent is mini-batch gradient descent. It computes the gradient for parts of the entire dataset, not the entire dataset:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{8}$$

Mini-batch gradient descent just uses n training examples in every gradient computation. But the most common choice is the stochastic gradient descent (SGD) which update a parameter for each training example $x^{(i)}$ and lable $y^{(i)}$:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{9}$$

SGD is much faster than the batch gradient descent and mini-batch gradient descent. Some optimization algorithms can be used to improve the performance of SGD. Momentum [17]adds a fraction $\gamma$ of the update vector of the previous step to the current update vector:

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta) \tag{10}$$

$$\theta = \theta - v_t \tag{11}$$

The most common choice of $\gamma$ is 0.9. Nesterov accelerated gradient (NAG) [16] is an improved version of Momentum by letting it know which direction the best is to move the parameter $\theta$:

$$v_t = \gamma v_{t-1} - \eta \nabla_\theta J(\theta - v_{t-1}) \tag{12}$$

Besides the algorithms above, Adagrad [9]adapts the learning rates to the parameters, update larger for infrequent parameters and smaller for frequent parameters. Then, there is no need to tune learning rate. But this algorithm may cause learning rate to shrink and go infinitesimal. Adadelta [19] can solve this problem by fixing the size of window of accumulated previous gradient. RMSprop [21]is a similar algorithm to Adadelta introduced by Geoffrey Hinton in his Coursea Class. Adam [9] is another method that computes adaptive learning rate for each parameter. AdaMax [9] is an extension of Adam. Nadam [8]combines RMPprop and momentum. For a great overview of some other common choices, refer to [13].

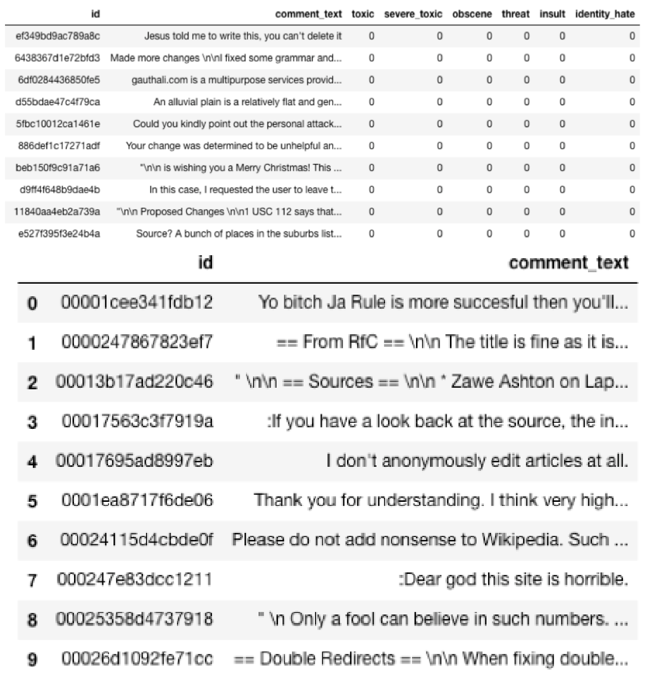
## 3 EXPERIMENT

### 3.1 Dataset

The dataset we use is from the competition Toxic Comment Classification Challenge on the Kaggle website [1]. The data consists of 159571 rows and 8 columns. Each row represents a comment. These 8 columns include the id number and content of each comment, and six types of toxicity. In reality, we can determine if a comment is toxic based on some kinds of key word in comment. The first 10 rows of the training and test data are shown in Figure 6. The type of comment_text is string line, toxic, severe_toxic, obscene, threat, insult, identity_hate are kinds of columns to judge the comment. 0 represents this comment doesn't contains such kinds of toxic words. 1 is vice versa.

### 3.2 Approach

After loading the essential packages such as Numpy, Pandas, and training and test data, we firstly clean the data. Then do the standard Keras preprocessing to let each comment in a list of equal length. Then, we start to define the model and fit the model. Lastly, we make a comparison between scores of predictions made by different models with different optimizer and dropout rates. In addition, we also add one more hidden layer to test the accuracy of the classification with best optimizer and best dropout rate we found.



| id | comment_text | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|---|
| ef349bd9ac789a8c | Jesus told me to write this, you can't delete it | 0 | 0 | 0 | 0 | 0 | 0 |
| 6438367d1e72bfd3 | Made more changes \n\nI fixed some grammar and... | 0 | 0 | 0 | 0 | 0 | 0 |
| 6df0284436850fe5 | gauthali.com is a multipurpose services provid... | 0 | 0 | 0 | 0 | 0 | 0 |
| d55bdae47c4f79ca | An alluvial plain is a relatively flat and gen... | 0 | 0 | 0 | 0 | 0 | 0 |
| 5fbc10012ca1461e | Could you kindly point out the personal attack... | 0 | 0 | 0 | 0 | 0 | 0 |
| 886def1c17271adf | Your change was determined to be unhelpful an... | 0 | 0 | 0 | 0 | 0 | 0 |
| beb150f9c91a71a6 | "\n\n is wishing you a Merry Christmas! This ... | 0 | 0 | 0 | 0 | 0 | 0 |
| d9ff4f648b9dae4b | In this case, I requested the user to leave t... | 0 | 0 | 0 | 0 | 0 | 0 |
| 11840aa4eb2a739a | "\n\n Proposed Changes \n\n1 USC 112 says that... | 0 | 0 | 0 | 0 | 0 | 0 |
| e527f395f3e24b4a | Source? A bunch of places in the suburbs list... | 0 | 0 | 0 | 0 | 0 | 0 |

|  | id | comment_text |
|---|---|---|
| 0 | 00001cee341fdb12 | Yo bitch Ja Rule is more succesful then you'll... |
| 1 | 0000247867823ef7 | == From RfC == \n\n The title is fine as it is... |
| 2 | 00013b17ad220c46 | " \n\n == Sources == \n\n * Zawe Ashton on Lap... |
| 3 | 00017563c3f7919a | :If you have a look back at the source, the in... |
| 4 | 00017695ad8997eb | I don't anonymously edit articles at all. |
| 5 | 0001ea8717f6de06 | Thank you for understanding. I think very high... |
| 6 | 00024115d4cbde0f | Please do not add nonsense to Wikipedia. Such ... |
| 7 | 000247e83dcc1211 | :Dear god this site is horrible. |
| 8 | 00025358d4737918 | " \n Only a fool can believe in such numbers. ... |
| 9 | 00026d1092fe71cc | == Double Redirects == \n\n When fixing double... |

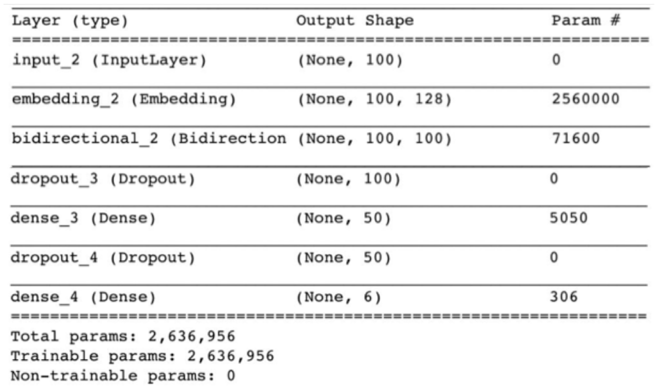
**Figure 6: Part of the training and test data.**

### 3.3 Cleaning the data and Keras preprocessing

Firstly, We replace missing values with unmeaning string. And then, using the functions in the text and sentence in the module preprocessing of Keras, we turn each comment into a list of equal length. Since the comment_text is string type, we use the built-in tokenizer in Keras to make it into token, which will be convenient in the following procedures.

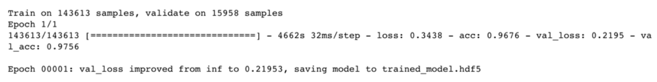
### 3.4 Define the model

We embedding the text into 128 length. We define the model using a function which contains all the parameters and its value. Because of the limited hardware we have, the number of epoch we set is just 2 and the size of batch we set is 32. We also change some parameters to compare scores and find the best one. Here, we use different optimizer and dropout rates to fit the model. In order to avoid overfitting, we use dropout. And because we use cross-validation instead of setting validation set directly, we use val_loss for parameter monitor. We use ReLU and sigmoid function as the activation functions of two hidden layers. Our network architecture is shown as Figure 7. We set the same dropout rates for both of the two hidden layers. In Figure 7, we set the dropout rate in 0.1, then in next part we change the dropout rate to test the accuracy of the model. The score submitted in Kaggle is shown in Table 2. We try 0.1, 0.2 and 0.3 to test the result. We also try three hidden layers, we set the dropout rate is 0.1, the optimizer is Nadam, and the dense_4 is 20, dense_5 is 6 (dense_5 is the parameter of the third hidden layer. Since each epoch took us a very long time to run. We can only test three hidden layers with limited resources. In addition, the result and score of one hidden layer with 0.1 dropout rate and Nadam optimizer is shown in Figure 8. The score in Kaggle with one hidden layer is 0.7238 which is much lower than two or three hidden layer. Figure 9 is the code of how we defined the model.



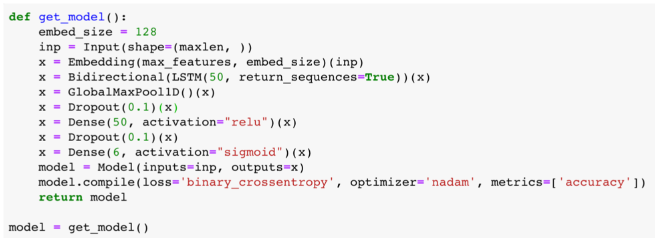
```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 100)               0
_________________________________________________________________
embedding_2 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 100, 100)          71600
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 306
=================================================================
Total params: 2,636,956
Trainable params: 2,636,956
Non-trainable params: 0
```

**Figure 7: Network architecture.**



```
Train on 143613 samples, validate on 15958 samples
Epoch 1/1
143613/143613 [==============================] - 4662s 32ms/step - loss: 0.3438 - acc: 0.9676 - val_loss: 0.2195 - va
l_acc: 0.9756

Epoch 00001: val_loss improved from inf to 0.21953, saving model to trained_model.hdf5
```

**Figure 8: One hidden layer.**



```
def get_model():
    embed_size = 128
    inp = Input(shape=(maxlen, ))
    x = Embedding(max_features, embed_size)(inp)
    x = Bidirectional(LSTM(50, return_sequences=True))(x)
    x = GlobalMaxPool1D()(x)
    x = Dropout(0.1)(x)
    x = Dense(50, activation="relu")(x)
    x = Dropout(0.1)(x)
    x = Dense(6, activation="sigmoid")(x)
    model = Model(inputs=inp, outputs=x)
    model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])
    return model

model = get_model()
```

**Figure 9: Model definition with two hidden layer.**

## 3.5 Fit the model

After defining the model, we fit the model based on the cleaned data. In our computer, the time of training each epoch is approximately half an hour. We let the validation_split be 0.1. There are two reasons for that. Firstly, we cannot choose leave-one-out cross-validation because of lack of enough computational resource. Secondly, score will decrease too much if we set ratio of split be 0.2 or even more, although high ratio of split will accelerate training.

## 3.6 Make prediction

Having the fitted model, we use test set to make prediction which is then appended to the sample submission file.

| Optimizer | Score |
|---|---|
| RMSprop | 0.9633 |
| SGD | 0.5989 |
| Adagrad | 0.9651 |
| Adadelta | 0.9596 |
| Adam | 0.9643 |
| Nadam | 0.9754 |

**Table 1: Score of prediction made by various model using different optimizers**

| Dropout Rate | Score |
|---|---|
| 0.1 | 0.9754 |
| 0.2 | 0.9747 |
| 0.3 | 0.9699 |

**Table 2: Score of prediction made by different dropout rate**

## 3.7 Obtain the score

Finally, we submit our .csv file to Kaggle and obtain the score we have. After trying many submissions, we can find the best one. Using different optimizer, we obtain scores for predictions made by different models. Our result for optimizer is shown in table 1. We also use the best optimizer we test which is Nadam, using different rate of dropout to test the accuracy of the model. The results with three hidden layers is in Figure 11. It's score is 0.9696 which is lower than 0.9754.

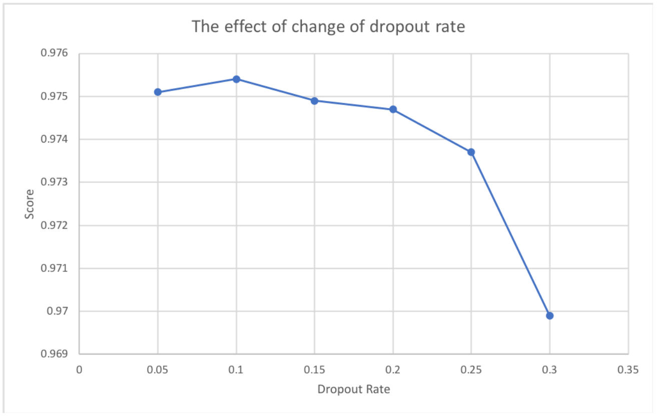
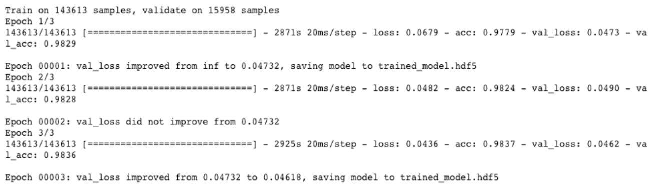


**Figure 10: The effect of 3 hidden layer.**

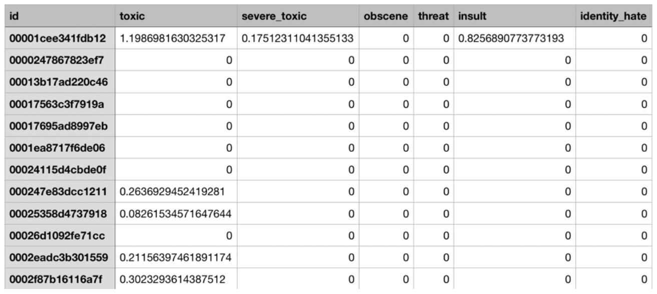


```
Train on 143613 samples, validate on 15958 samples
Epoch 1/3
143613/143613 [==============================] - 2871s 20ms/step - loss: 0.0679 - acc: 0.9779 - val_loss: 0.0473 - va
l_acc: 0.9829

Epoch 00001: val_loss improved from inf to 0.04732, saving model to trained_model.hdf5
Epoch 2/3
143613/143613 [==============================] - 2871s 20ms/step - loss: 0.0482 - acc: 0.9824 - val_loss: 0.0490 - va
l_acc: 0.9828

Epoch 00002: val_loss did not improve from 0.04732
Epoch 3/3
143613/143613 [==============================] - 2925s 20ms/step - loss: 0.0436 - acc: 0.9837 - val_loss: 0.0462 - va
l_acc: 0.9836

Epoch 00003: val_loss improved from 0.04732 to 0.04618, saving model to trained_model.hdf5
```

**Figure 11: The result of prediction with 3 hidden layers.**

Besides the optimizer, we also modify the dropout rate to fit the model. Our result for different dropout rate while using Nadam as our optimizer is in figure 4.

From the table 1, we find that the Nadam is the best optimizer for our task. For the dropout rate, we find that final score increases if dropout rate increases from 0.05 to 0.10, but score decreases as dropout rate continues to increase. The reason might be high dropout rate may cause network to ignore too much useful information, which would cause accuracy loss. Part of the result is shown in Figure 12. This file is what we submit to the Kaggle.



| id | toxic | severe_toxic | obscene | threat | insult | identity_hate |
|---|---|---|---|---|---|---|
| 00001cee341fdb12 | 1.1986981630325317 | 0.17512311041355133 | 0 | 0 | 0.8256890773773193 | 0 |
| 0000247867823ef7 | 0 | 0 | 0 | 0 | 0 | 0 |
| 00013b17ad220c46 | 0 | 0 | 0 | 0 | 0 | 0 |
| 00017563c3f7919a | 0 | 0 | 0 | 0 | 0 | 0 |
| 00017695ad8997eb | 0 | 0 | 0 | 0 | 0 | 0 |
| 0001ea8717f6de06 | 0 | 0 | 0 | 0 | 0 | 0 |
| 00024115d4cbde0f | 0 | 0 | 0 | 0 | 0 | 0 |
| 000247e83dcc1211 | 0.2636929452419281 | 0 | 0 | 0 | 0 | 0 |
| 00025358d4737918 | 0.08261534571647644 | 0 | 0 | 0 | 0 | 0 |
| 00026d1092fe71cc | 0 | 0 | 0 | 0 | 0 | 0 |
| 0002eadc3b301559 | 0.21156397461891174 | 0 | 0 | 0 | 0 | 0 |
| 0002f87b16116a7f | 0.3023293614387512 | 0 | 0 | 0 | 0 | 0 |

**Figure 12: Classification result.**

## 3.8 Future work

Deep learning has made huge progress in the last few years. The most significant progress deep learning has made is in computer vision and natural language processing. Convolutional neural network (CNN) was introduced by LeCun et al. [12], and was used very commonly in the computer vision community. The reason why CNN is popular in computer vision can be divided into four parts: local connections, shared weights, pooling and the use of many layers [12]. Besides CNN, recurrent neural network (RNN) is another popular network, especially in the natural language processing community. Because of its ability to handle the sequential input, such as speech and language, it is much better than other networks. But most of the progress we have made focus on supervised learning. In fact, human learning is largely unsupervised: we are always discovering the structure of object through observing it, instead by being told its name. Breakthrough in unsupervised learning can be expected in the near future. With the development of hardware, software and algorithms, artificial intelligence will embrace the bright future.

## 4 CONCLUSION

In this task, we use the data from Kaggle website, to fit the model with different optimizer and parameters, obtain scores and compare them. We find the best model and get as high score as we can. We found that bidirectional LSTM works well in this task, the reason for that may be the information of the first few parts of comment is highly correlated with the final part. Some techniques, like regularization and dropout, can be applied to avoid overfitting. Choices of optimizer may have significant effect of the final score. We think this is similar with the result in [20]. This result is also interesting because we can make use of this idea to classify the review in Amazon, Airbnb, Yelp and so on.

## REFERENCES

[1] [n. d.]. https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[2] Graves A. and Schmidhuber J. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18 (2005), 602–âĂŞ610.

[3] MartÃŋn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, and Lukasz Kaiser. 2015. TensorFlow: Large-Scale Machine Learning on Heerogeneous Distribution Systems. (2015). https://www.tensorflow.org/

[4] Y. Bengio, R. Ducharme, and P. Vincent. 2003. A neural probabilistic language model. (2003).

[5] FranÃğois Chollet and Others. 2015. Keras. (2015). https://github.com/fchollet/keras

[6] Church, K.W., and Hanks. P. 1990. Word Association Norms, Mutual Information and Lexicography, computational Linguistics. *Word Association Norms, Mutual Information and Lexicography, computational Linguistics* (1990).

[7] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for information science* (1990).

[8] Timothy Dozat. 2016. Incorporating Nesterov Momentum into Adam. *ICLR Workshop* 1 (2016), 2013–2016.

[9] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[10] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation* 9 (1997), 1735–1780.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. (2014). https://doi.org/DOI:http://dx.doi.org/10.1145/2647868.2654889

[12] Y. LeCun, Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W., and Jackel L.D. 1990. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems* 2 (1990), 396–404.

[13] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert MÃijller. Efficient BackProp. 1998. (1998), 9–50.

[14] Levy, O. Goldberg, and Y. Dagan. 2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics* 3 (2015), 211–255.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. The patent holder's dilemma: Buy, sell, or troll? *In ICLR Workshop papers* (2013).

[16] Yurii Nesterov. [n. d.]. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). *Doklady ANSSSR* 269 ([n. d.]), 543–547.

[17] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society* 12 (1999), 145–151.

[18] X. Rong. 2014. word2vec parameter learning explained. *arXiv preprint arxiv* (2014).

[19] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. 2015. Evaluation methods for unsupervised word embeddings. *In Proc. of EMNLP* (2015).

[20] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, Salakhutdinov, and Ruslan. 2014. Dropout: A simple way to prevent neural networks from overfitting. 15 (2014), 1929–1958.

[21] T. Tieleman and G Hinton. 2012. Lecture 6.5âĂŤRmsProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* (2012).

[22] Peter D. Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *In Journal of Artificial Intelligence Research* 37 (2010), 141–188.