Toxic Comment Classification by Regression and Neural Networks

CSE802 Project Proposal

Zhuangdi Zhu Wei Wang Zhiming Jiang

Department of computer Science and Engineering

Michigan State University

East Lansing, Michigan, USA

{zhuzhuan, wangwe90, jiangzh7}@msu.edu

*Abstract*— In this project, we proposed approaches to automatically recognize toxic comments on social networks. Digital abuse is a critical issue worldwide. The approach of manually rating toxic comments works with a huge cost. Therefore, we propose classification models that can automatically detect different types of toxicity like threats, obscenity, insults, etc, using pattern recognition approaches. We use logistic regression as well as neural networks techniques to build different models and conducted evaluations to compare the performance of them.

*Keywords— neural networks; natural language processing, regression;*

# Introduction

### Motivation

In this project, we aim to provide a tool which recognizes toxic comments presented on social networks. Digital abuse is a critical issue worldwide. Due to the prevalence of social platforms such as YouTube and Twitter, people have more channels to express their opinions on the internet, for better or for worse. On one hand, the anonymity feature of the network leave people less constrained when making comments. Some people may make reckless comments without taking any consequences. On the other hand, some people may give inappropriate comments even without awareness because of the culture difference or misunderstanding.

The abuse of toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion) on the internet can be harmful to both individuals and the society as a whole. For individuals, the threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Victims of severe digital abuse may even end up with depression, taking the loss of reputation or privacy. For companies, they have to struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

Depriving users’ rights of expressing comments is not the right solution to digital abuse. Many online platforms and communities are hiring labors to manually rate toxic comments in order to take reactions. However, the manual rating approach works at large cost. First, manual detection of toxic comments is time consuming and non-efficient. The burst of negative events on the internet can generate large amount of information in a short time, which overwhelms the human raters. Besides time, money spent on paying toxic comment raters is also a huge expenditure. Moreover, being exposed to toxic comments hours and days leave human raters under big pressure. All of these issues can be addressed if we build a detection model which automatically detects toxic comment.

### Literature Review

Our work belongs to the field of sentiment analysis, which can be considered as an integration of natural language processing and pattern recognition. Much work can be found along these two lines [8-14]. In this project, we focus on the approaches used for feature extraction and model training.

#### Regression approaches for sentiment analysis

喵喵喵

#### Deep learning approaches for sentiment analysis

With the blossom of large amount of corpus available on the internet, deep learning approaches using neural networks have earned increasing popularity the field of Sentiment. Researchers from Stanford University have proposed complex neural network models to conduct sentence-level sentiment analysis. An example is the Matrix-Vector Recursive Neural Network (MV-RNN), proposed by Socher *et al* [8]*.* This model takes input a parse tree of a sentence and generates the sentiment category of the root node of the parse tree by recursively building it in a bottom-up fashion.

Chung *et.al* compares different types of recurrent units in recurrent neural networks [9]. They found that more sophisticated units with a gating mechanism, such as a long short-term memory (LSTM) unit and a recently proposed gated recurrent unit (GRU) performs better when modeling long sequence data. Based on traditional GRU, Zhou et. al proposed a novel recurrent cell called Minimal Gate Unit (MGU) [10]. All of the above work inspires us to leverage recurrent neural network to build a comment classifier.

### Problem Statement

In this project, we aim to build a multi-headed model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based. Especially, we abstract the problem as a multi-label classification problem: given a text , we want to predict the probability that this text belongs to certain toxicity types:

Here, is a text composed of English words; is a probability vector of dimension , as we have possible toxicity types; the value in each dimension of indicates the likelihood that this text belongs to a certain toxicity type. We are given a dataset of which is a collection of labeled texts. For each sample , we have a label , which is a binary vector of dimension .

# Methedologies

This project involves four stepwise: data collection & pre-processing, feature extraction, model training, and the performance evaluation.

## Data preprocessing

The dataset we use is from Wikipedia’s talk page edits. It contains a large number of Wikipedia comments which have been labeled by human raters for toxic behavior. The dataset is available online [1]. It contains around 160, 000 samples, each sample being a comment written by English. A sample is associated with six types of toxicity:

|  |  |
| --- | --- |
|  |  |

Each toxicity type is labeled as either 1 or 0. It worth noting that a comment can be labeled with more than one toxicity type. We take two steps to pre-process the dataset. First, for each text in the dataset, we clean it by removing some stopping words, special characters and punctuations. Stopping words are frequently used words such as *the, an, a, will, shall*, etc. Next, we tokenize each text into a list of words, so that we can get the vocabulary of the whole dataset, where each word in the vocabulary is a one-bit-hot vector: , so that only one bit in is one, and the dimension of this vector is vocabulary size.

Before we extract features from this dataset, we conducted a word-frequency analysis. We separate the training dataset into two groups: the neutral subset and the negative subset. The neutral subset contains all texts that are not labeled as any toxicity type, while negative subset contains texts that are labeled as at least one toxicity type. We use a graphical tool [2] to show the visualization of the neutral group in Figure 1. The visualization of the negative group is not shown because it may bring discomfort to our readers.



1. Visualization of the netural dataset

## Feature Extraction

In this section, we introduce three different approaches used for extracting features: count vectorization and TF-IDF vectorization approaches for training a regression model, and word embedding approaches for training neural network models.

### Count Vectorization

Count vectorization is a procedure that calculates the occurrence(frequency) of a particular token appeared in one document normalized by all tokens across all documents. Token is learned when taking all documents into account. Each individual token occurrence is treated as a feature; the vector containing all token frequencies for a given document is used to train classifier model.

### Tf-idf Vectorization

Tf-idf (term frequency-inverse document frequency) [4], is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf weight is often used in information retrieval and text mining. The importance of a word is related with the appearance and the frequency in the document. Tf-idf is often used by searching engines in scoring and ranking a document’s relevance given a user query [5].

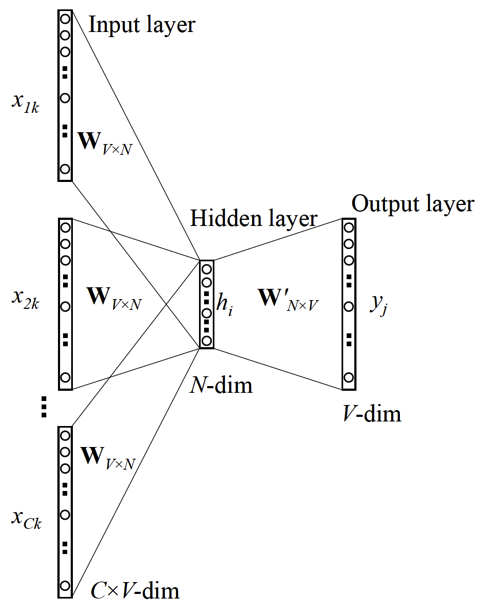
The tf-idf weight contains two parts, the normalized term frequency and the inverse document frequency. For a document and a word , tf-idf is calculated as following:

### Word to Vector Representation

In order to generate trainable features, we need to map each word into a vector representation. Initially, a straight-forward approach to represent words is using one-hot vector. So that each word is an vector with all 0s, except one bit set to 1 to specify the index of that word in the sorted vocabulary, and |V| is the size of our vocabulary. This notation represents each word as a completely independent entity, without giving any notion of semantic similarity.

In order to improve this approach, we can map one-hot vectors to a smaller subspace which can encodes the semantic relationships between words, and this kind of mapping is called word-embedding. Word Embedding represent words in a continuous vector space where semantically similar words are mapped to nearby points. In another word, Word Embedding is a predictive model that predicts the occurrence of a certain word given some context information. It comes in two flavors: The Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. The main difference between CBOW and Skip-Gram model is that, CBOW predicts a center word from the surrounding context, while Skip-Gram: does the inverse and predicts surrounding context words given a center word. In our project, we adopt the CBOW approach and implement it using the *Keras* python module.







1. CBOW versus Skip-Gram Embedding.

## Model Training

In this section, we first designed and implemented three kinds of classifiers: a gradient-boost based classifier, a CNN based classifier, and a RNN based classifier. Next, we also combine the CNN and RNN model into an integrated network, in the hope of getting better performance than a single RNN or CNN module. We will compare the performance of each classification model in the Evaluation section.

### XGBoost Classifier

XGBoost (Extreme Gradient Boosting), is an optimized distributed gradient boosting model. It provides a parallel tree boosting framework, using gradient boosting to speed up machine learning algorithms. We plan to use logistic regression classifier in XGBoost.

Logistic regression is widely used on natural language processing, for example, in topic classification, text categorization, detection of spam pages. Logistic regression measures the relationship between an output variable (category) and one or more independent variables, which are usually continuous, by using probability scores as the predicted values of the dependent variable.

For input feature vectors , outcome ranges between 0 and 1 which represents the prediction of probability. For probability greater than 0.5, we would classify the observation as positive, otherwise, as negative. The probability of the out is assumed to follow a parametric model given by the sigmoid function:

### CNN-based classifier

Conventionally, CNN is widely applied to applications like image classification or object detection. We plan to leverage CNN for text classification for two reasons: first, we think that the important thing for toxicity content classification is to find sentiment features, like abuses or identity-hates, which are usually related one or two single words and can be extracted using a CNN. Second, we think that CNN converges faster than RNN due to its simple structure. Since each sample in the dataset is a short comment instead of a long paragraph/document, we can easily use padding approach to make it fit for a CNN classifier.

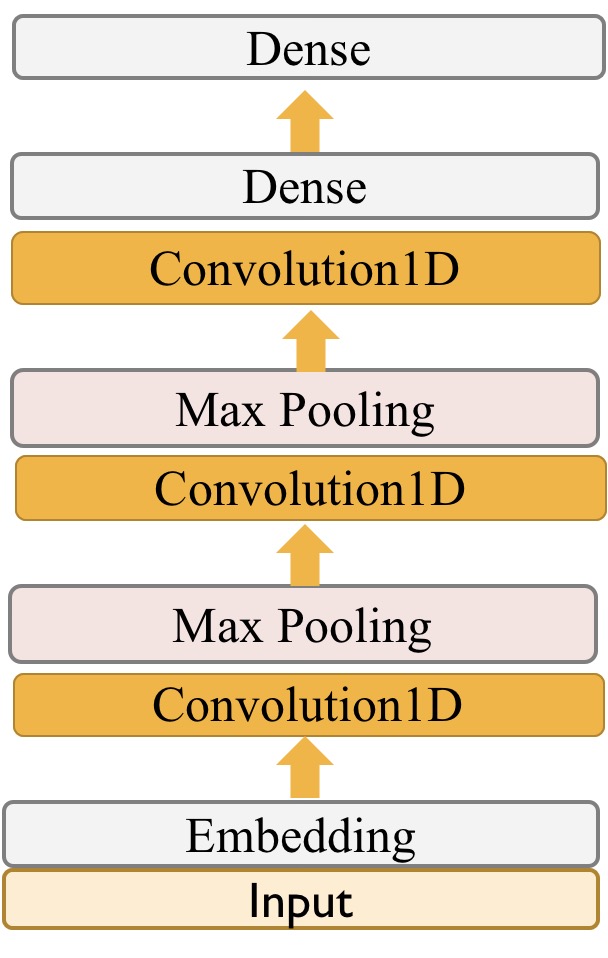
As we know, the major part of CNN is constructed by the convolutional layer, the pooling layer, and the fully-connected Layer.

The convolution layer is the building block of a CNN model. In this layer, a kernel matrix is used to conduct a dot production on each ‘slice’ of the training sample in order to extract features and filter out useless information. You can think this kernel matrix as a sliding window which traverses through the training sample, so that after the convolution layer, you will have a feature vector (or matrix, based on the structure of yur training sample) that can be used for further training.

The pooling layer is used to remove local variance and decrease the dimension of the feature vector, which therefore alleviates the problem of overfitting. There are two basic pooling approaches: max pooling and average pooling. We will explore each pooling approach and pick the optimal one in our following experiments.

After layers of convolution and pooling, we need activation functions to bring the needed non-linearity property. For the active function, we plan to use the classic one which is Relu [7]. Finally, we have a single vector which we can use for Fully connected layers.

In practice, we plan to use Python-Keras language to establish the CNN (convolution neural network) classifier. The model structure is given in Fig. 3.

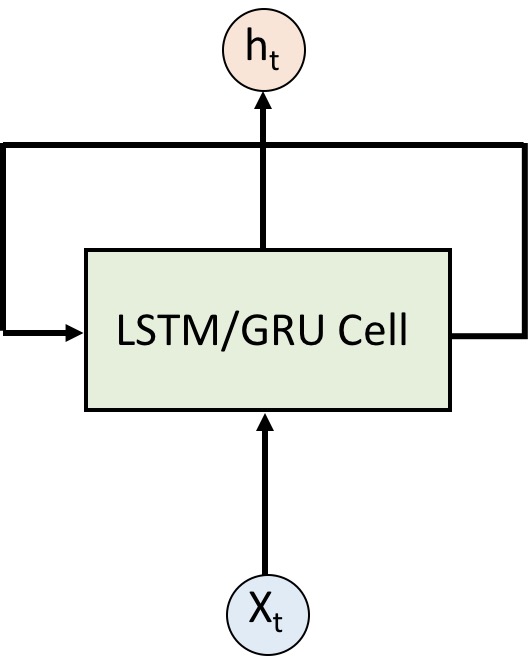


1. CNN-based model

### RNN-based Network Classifier

RNN is widely applied in many areas such as protein action recognition, speech recognition, machine translation and handwriting generation. It is a recurrent model that can learn order dependence in sequence prediction problems. As shown in Fig 3. In an RNN model, a chunk of neural network cells, looks at some input and outputs a value . A loop allows information to be passed from one step of the network to the next.

In this project, we explore three RNN variants: a LSTM network, a GRU network, and a Bidirectional RNN network, and show how we build a RNN-based model using these three variants.



1. RNN

#### RNN varitant 1: LSTM

Long Short-Term Memory (LSTM) is an variant of the vanilla RNN [7]. It differs from the standard RNN by having a new structure called memory cell. The memory cell is composed of four main elements: an input gate, a forget gate, an output gate, and a neuron with a self-recurrent connection (a connection to itself). The three gates contained in the memory cell can be represented as follows:

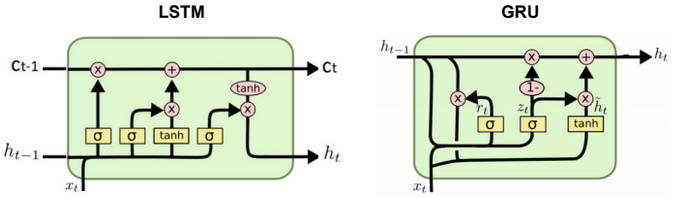
Where Eq. (1) – (3) represents the input gate, forget gate, and the output gate, respectively. Then the updated memory cell contains information both from the three gates and its original cell information:

The memory cell then conveys information to the updated hidden state:

Finally, the output layer looks like following:

#### RNN variant 2: GRU

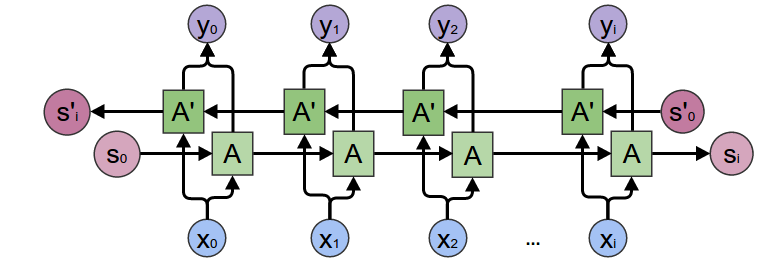
GRU network is also a recurrent neural network with gated cells, where each of the cell is called Gated Recurrent Unit (GRU). The idea behind a GRU is quite similar to that of aLSTM*.* Different from LSTM units, a GRU unit does not have an output gate.We show the cell structure of LSTM and GRU in Fig. 4.



1. LSTM cell versus GRU cell

#### RNN variant 3: Bidirectional RNN

The structure of a Bidirectional RNN is shown in Fig. 5. It is a combination two independent RNNs. The input sequence is fed in normal time order for one network, and in reverse time order for another. The outputs of the two networks are combined at each step, with the common combination being concatenation or summation. With this structure, Bidirectional RNN allows the networks to have both backward and forward information about the sequence at every time step.



1. Bidirectional RNN

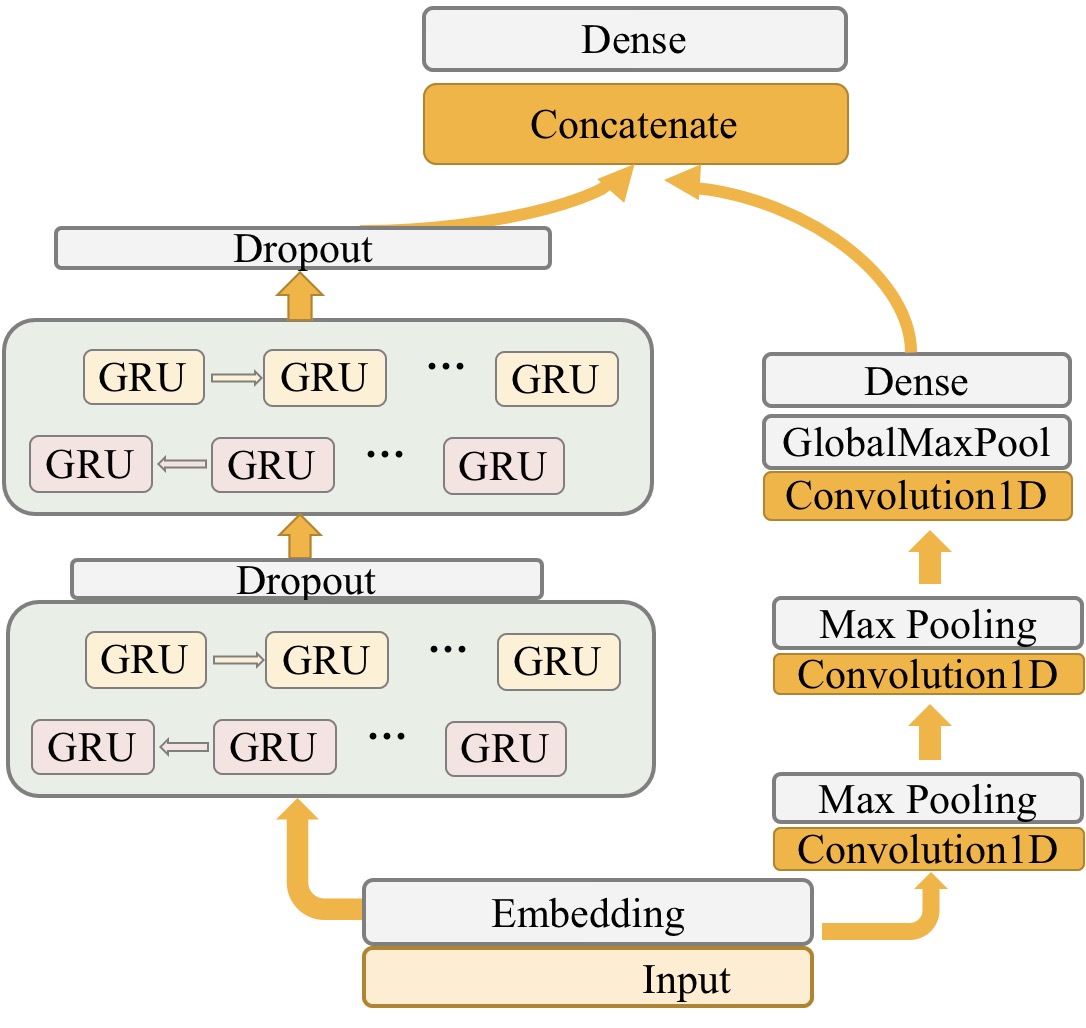
Based on the abovementioned RNN variants, we designed a bidirectional RNN models using GRU cells. It contains two bidirectional RNN layers, with each layer containing 64 hidden cells. The final output layer is a dense layer with 6 units, as we have six toxicity types.



1. Bidirectional RNN model using GRU cells

#### Integrated model using both RNN and CNN

Given the CNN based model and RNN based model, we propose an integrated network by merging the two models together, so the output feature map of the CNN model and the RNN model will be concatenated and fed into the final output layer. The structure of this integrated model is shown in Fig. 7.



1. Integrated model using both RNN and CNN

## Performance Evaluation

We apply different on a real-word dataset collected by Wikipedia’s talk page edits. In this section, we will first introduce the metrics used for evaluation the model performance. Next, we evaluate the performance of each model. Finally, we compare the performance across different models.

### Metrics

We adopt four metrics to evaluate the models’ performance:

* Training accuracy
* Training loss
* Testing accuracy
* Testing loss

Especially, for loss function, we adopt binary cross entropy loss. Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. A perfect model would have a log loss of 0. Binary cross-entropy can be calculated as follows:

Besides the above metrics, we also use ROC curve (receiver operating characteristic curve) and AUC (area under curve) [6] to evaluate the performance of the regression model. ROC curve illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold. For a binary classification problem. One of two labels positive or negative is assigned to the input. There are four possible outcomes from a binary classifier. If the outcome from a prediction is positive and the actual value is also positive, then it is called a true positive (TP); if the actual value is negative then it is a false positive (FP). If the outcome from a prediction is negative and the actual value is also negative, then it is called a true negative (TN); if the actual value is positive then it is a false negative (FN). TPR and FPR are calculated as following:

AUC is the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. AUC varies between 0 and 1, an uninformative classifier result in AUC of 0.5. Better classifier results in higher AUC.

### Regression model evaluation

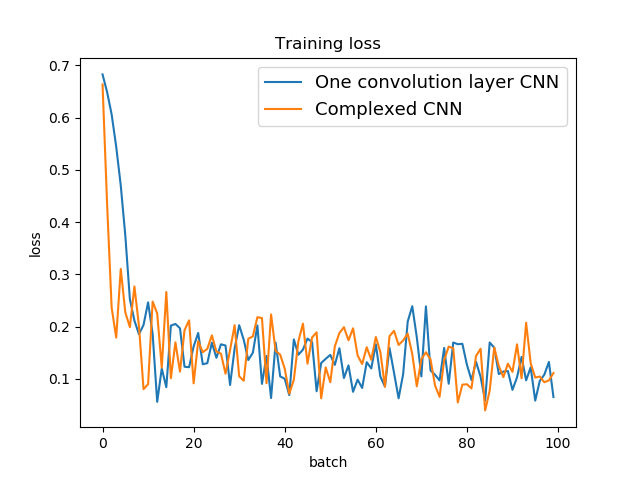
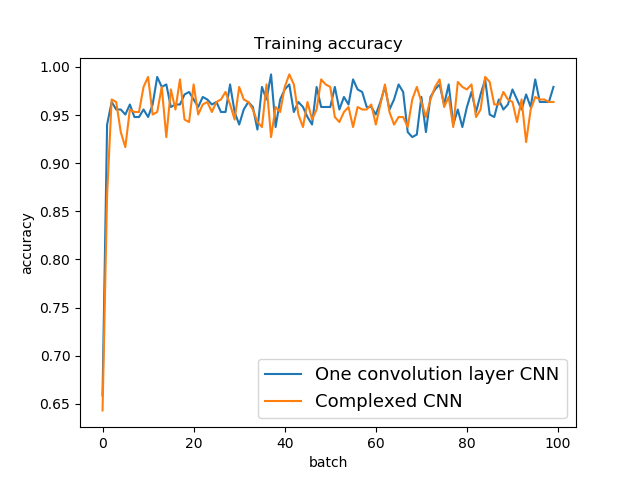
喵喵喵

### CNN model evaluation

We evaluated our RNN model using the Wikipedia comments dataset. Please be noted that due to the large amounts of training samples available (160,000 samples), our models converge quickly after a few of epochs. Therefore, all of the following experiments are done with epoch = 1, batch size = 64.

For the CNN based model, we compare out model’s performance with one benchmark model, which is a CNN model with one convolution layer and one max-pooling layer. The convolution layer is in size, followed by a rectified linear unit (*relu*) activation layer.

We show the performance comparison in Table 1. We can see that our model outperforms the benchmark in training accuracy/loss and testing loss. For the testing accuracy, they achieve the same result. We also compare the convergence speed of these two models in Fig. 9. We can see that our model, although with a more complexed network structure, converges faster than the simpler version in terms of training loss, and achieves almost the same pace in terms of training accuracy.



1. Training performance for CNN-based models

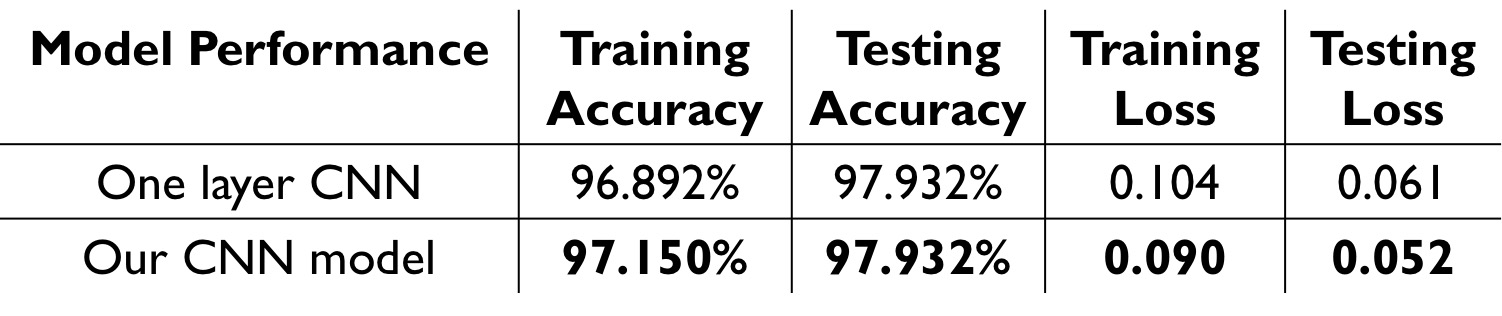
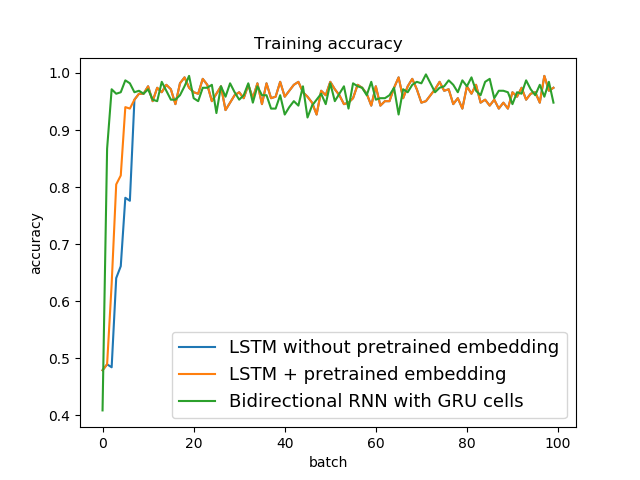
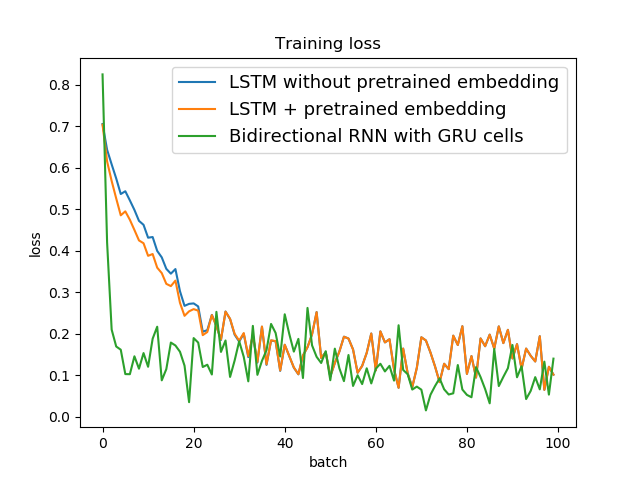


Table. 1: Evaluation results for RNN based model

### RNN model evaluation

We compare our model’s performance against two benchmarks: the first benchmark is a standard LSTM network, i.e. a network with one regular RNN layer. The second is a LSTM network with a pre-trained embedding matrix. We show the training accuracy and training loss for the first 100 batches in Fig. 10. It can be observed that a pre-trained embedding matrix does improve the training performance with faster convergence speed. It can also be observed that our model, a bidirectional 2-layer RNN model using GRU cells, outperforms the two benchmarks with lower loss, higher accuracy, and faster convergence speed. Table 2 shows the detailed performance of each of the three models.





1. Training performance for RNN-based models

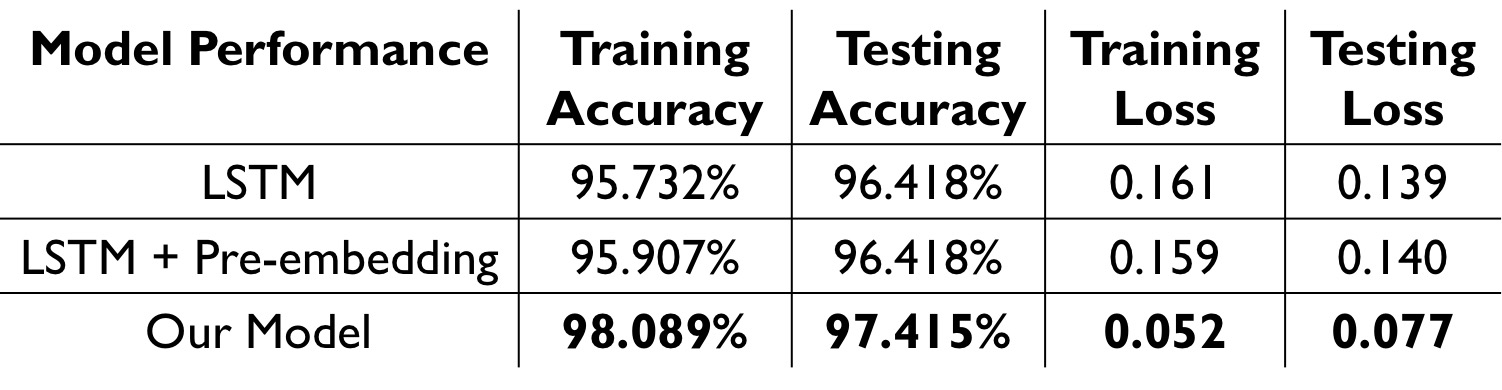


Table. 2: Evaluation results for RNN based model

### Integrated model evaluation

# Contributions of teammembers

In this project, we together designed, implemented, and evaluated different models wrote this report project together.

* + - 1. Wei Wang designed and implemented the regression-based model. She is also responsible for the feature extraction, model training, and evaluation of this model. She implemented the TF-IDF vectorization and word-count vectorization approaches for feature extraction. She also implemented the evaluation using those metrics used for the regression model.
      2. Zhuangdi Zhu designed and implemented the RNN-based model, as well as the integrated model using both RNN and CNN structure. She is responsible for the feature extraction, model training and evaluation of these two models. She wrote the Python/Matlab code used for plotting the evaluation results for the neural network part.
      3. Zhiming Jiang designed and implemented the CNN-based models. He is responsible for the feature extraction, model training and evaluation of the CNN model. He helped Zhuangdi on pre-processing the data for feeding neural networks, as well as model evaluation.

##### References

1. Data link for Kaggle competition: toxic comment classification: <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data>
2. Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016.
3. Salton, Gerard, and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." *Information processing & management* 24.5 (1988): 513-523.
4. Ramos, Juan. "Using tf-idf to determine word relevance in document queries." *Proceedings of the first instructional conference on machine learning*. Vol. 242. 2003.
5. Hanley, James A., and Barbara J. McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology* 143.1 (1982): 29-36.
6. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
7. Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.
8. R. Socher, B. Huval, C. D. Manning, and A. Y. Ng. "Semantic compositionality through recursive matrix-vector spaces. " In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning,* pages 1201–1211. Association for Computational Linguistics, 2012.
9. Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:*1412.3555 (2014)
10. Zhou, Guo-Bing, et al. "Minimal gated unit for recurrent neural networks." *International Journal of Automation and Computing* 13.3 (2016): 226-234.
11. Salton, Gerard, and Christopher Buckley. "Term-weighting approaches in automatic text retrieval." Information processing & management 24.5 (1988) : 513-523.
12. Nasukawa, Tetsuya, and Jeonghee Yi. "Sentiment analysis: Capturing favorability using natural language processing." Proceedings of the 2nd international conference on Knowledge capture. ACM, 2003
13. Liu, Bing, and Lei Zhang. "A survey of opinion mining and sentiment analysis." Mining text data. Springer US, 2012. 415-463.
14. Cambria, Erik, et al. "New avenues in opinion mining and sentiment analysis." IEEE Intelligent Systems 28.2 (2013): 15-21.