Toxic Comment Classification Challenge – Identify and classify toxic online comments

Ruixiao Duan

I. Introduction

TOXIC comment classification has draw intension recently mainly due to the overall volume of comments generated through websites, such as Wikipedia, Amazon.com etc. Millions of users have unrestricted access to vast amounts of content that allows for privileges unimaginable several decades ago, such as access to knowledge bases or latest news within just a few clicks. However, due to Internets non-restrictive nature and anonymity, some users misuse Internet to promote offensive and hateful language, which contaminate experiences of regular users, affects business of online companies, and may even have severe real-life consequences. To mitigate these detrimental effects, many companies (including Yahoo, Facebook, and YouTube) strictly prohibit hate speech on websites they own and operate, and implement algorithmic solutions to discern hateful content. Yet, scale and multifacetedness of the task renders it a difficult endeavor, and hate speech still remains a problem in online user comments. Based on that, we participate Jigsaw sponsored online competition on Kaggle, namely *Toxic Comment Classification Challenge*.

II. Related work

The prior arts mainly fall into two categories – traditional statistical machine learning method in text classification and deep-learning enabled text classification, as we will enumerate below.

A. Traditional statistical machine learning method

In this vein, the prior works are mostly published before 2012. People proposed a modified logistic regression to approximate the optimization of SVM by a sequence of unconstrained optimization problems. The authors then prove that the approximation will converge to SVM, and propose an iterative algorithm called MLR-CG which uses Conjugate Gradient as its inner loop. Experimental results demonstrates that on Reuters Corpus Volume I (RCV1) dataset which contains 781,265 training samples, 23,149 testing samples, 4 (binary) classes with 500 features per sample, the proposed algorithm achieved approximately the same (87.17% compared to 87.21% from SVM on test set accuracy) with reduced-complexity.

Moreover, others employed Expectation-Maximization method on text classification. The introduced algorithm is able to learn both labeled and unlabeled documents based on the combination of EM algorithm and a naive Bayes classifier. The algorithm first trains a classifier using the available labeled documents, and probabilistically labels the unlabeled documents. It then trains a new classifier using the labels for all the documents, and iterates to convergence. Experimental results, obtained using text from three different real-world tasks, show that the use of unlabeled data reduces classification error by up to 30%.

SVM has also been utilized to classify text, someone introduced a new algorithm for performing active learning with support vector machines, i.e., an algorithm for choosing which instances to request next. The authors' experimental results show that employing the proposed active learning method can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.

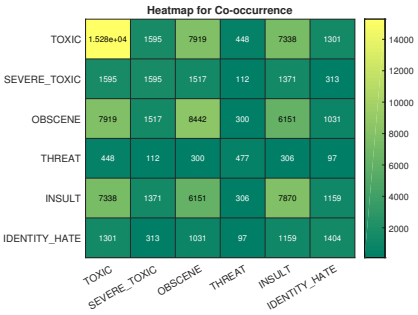


Fig. 1: Co-occurrence plot.

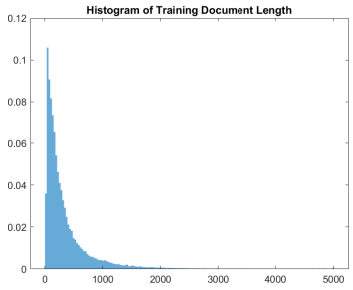


Fig. 2: Histogram of comment text length, all categories.

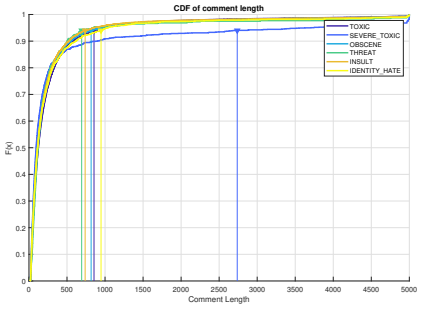


Fig. 3: CDF plot of comment text length, malicious categories.

B. Deep-learning based text classification method

With the computation power raises drastically recent years, the emergence of deep-layer neural networks (deep learning) starts dominate the domain of text classifications.People proposed character-level convolutional neural networks.

Recurrent neural networks (RNN) can also be utilized to do text classification, people compared several neural network based methods upon different datasets. We can conclude that there is no such model yet to outperform the others, i.e., there is no free lunch.

Some variations in sequential models can also be used here, e.g., Long Short Term Memory (LSTM), Gated Recurrent Units (GRU).Someone proposed a new framework named C-LSTM, who utilizes CNN to extract a sequence of higher-level phrase representations, and are fed into a LSTM to obtain the sentence representation. The authors claims that C-LSTM is able to capture both local features of phrases as well as global and temporal sentence semantics.

III. Project Perspectives

In this project, we are going to try different classification techniques, mainly fall into three categories, logistic regression methods, support vector machine, and deep-learning based mechanisms. Regarding the criteria in this competition, Kaggle calculates the mean columnwise Area Under the Curve (AUC), where the curve is Receiver Operating Characteristic (ROC) curve.

In our first part of experiments, which is based on traditional methods, we would like to try logistic regression classification method and then submit the results to the Kaggle website to get evaluated. In the implementation, we would like to employ scikit-learn Python packages for legacy methods. In the second part of experiments, we would like to apply SVM also in scikit-learn package. Finally, we utilize Keras to implement sequential models, for example, LSTM and GRU.

Here are some steps we need to follow,

- Text overview
- Word vectorization
- Model construction
- Training
- Prediction

IV. Dataset overview

In this section, we would like to firstly take a look at the dataset provided by this competition.

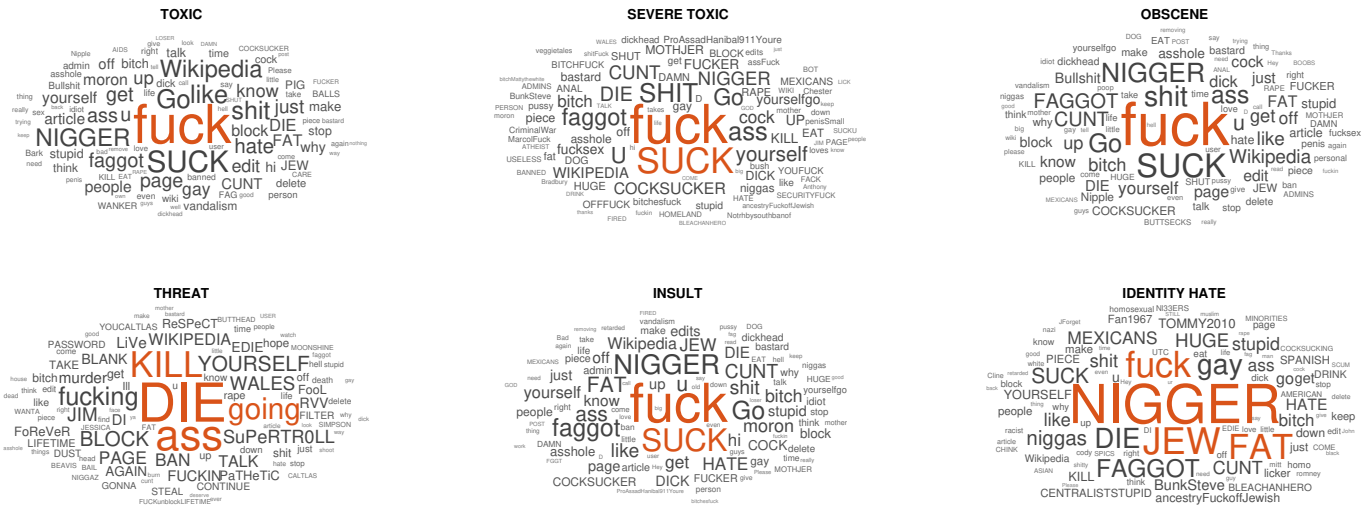


Fig. 4: Word-cloud for malicious categories

A. *Occurrences*

The majority of comments (150k samples in training set) are benign (or neutral) comments. Out of the complete dataset, 9.64% of them are toxic, 1.01% of them are severe toxic, 5.33% of them are obscene, 0.32% of them are threatening, 4.97% of them are insulting, and 0.85% of them composes identity hatred inclinations. Fig. 1 shows the co-occurrence, in the dataset. Interestingly but not surprisingly, severe toxic comments are all toxic comments, by nature.

B. *Comment text length selection*

In this section, we would like to study sentence-level characteristics of training set. According to Fig. 2 and Fig. 3, the majority (95 percentile) of the comments text length are less than 1000 ASCII characters. In the CDF plot of the malicious categories, we observe that there are some exceptions, `severe toxic` comments tends to be longer than the others.

C. *Word level analysis*

In Fig. 4, as we can see, the majority of the words are malicious. Yet interestingly, *Wikipedia* under the `toxic` comments, *mothjer* under the `severe toxic` categories, are worth noticing. Seemingly neutral words, e.g., *Wikipedia*, shows up as a majority (observed by word font-size, proportional to occurrence), and mis-spelled words like *mothjer* and their variations shall also be removed, and they may not appear in any pre-trained word embeddings (stated below).

V. Data vectorization for statistical machine learning

Before model construction, we need to convert text data to a feature matrix with m training samples and n features.

Here, we use *TF-IDF vectorizer* to convert a collection of raw document to a matrix of TF-IDF features. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection. The *TF-IDF* value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the collection that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

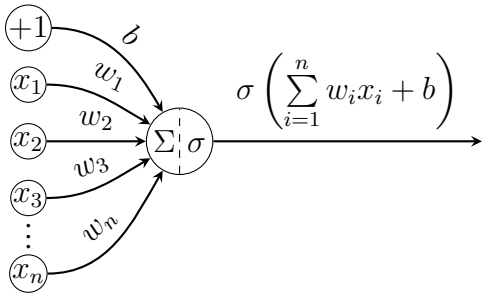


Fig. 5: Logistic regression

TF is *term frequency*, meaning the appearance of a specific word in this sentence. While *IDF* is *inverse document frequency* which means inverse of a ratio that number of documents with this word to total number of documents. The formula is given below for word *w* in document *D*,

$$TF_{wD} = \frac{\text{number of times } w \text{ appears in document } D}{\text{total number of words in document } D} \tag{1}$$

$$IDF_w = \log \frac{\text{total number of documents}}{\text{number of documents with word } w + 1} \tag{2}$$

Finally the weight *TF-IDF* is given by,

$$TF\text{-}IDF_{wD} = TF_{wD} \cdot IDF_w \tag{3}$$

In this project, we need to replace document by sentence, which means for each sentence in text data, we are supposed to calculate the value of *TF-IDF* for each word.

Here are some steps we need to follow in preprocessing:

- Define a word bag to cover all words appearing in dataset.
- For each sentence in dataset, compute *TF-IDF* value for each word in word bag, and append the numbers in a row vector.
- Repeat previous step for all sentence and form a matrix with number of sentences in row and size of word bag in column.

In implementation, there is a function in `sklearn` package.

VI. Statistical machine learning methods – logistic regression

In a nutshell, logistic regression can be represented by a single perceptron, i.e. in Fig. 5

A. Forward Propagation

In the forward propagation step, the algorithm basically calculates, for a n features, m total sample training dataset, input $\mathbf{X} \in \mathbb{R}^{m \times n}$, output $\mathbf{Y} \in \mathbb{R}^{m \times 1}$,

$$\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{b} \tag{4}$$

$$\hat{\mathbf{Y}} = \frac{1}{1 + e^{-\mathbf{Z}}} \tag{5}$$

where $\mathbf{Z}, \mathbf{b} \in \mathbb{R}^{m \times 1}, \mathbf{W} \in \mathbb{R}^{n \times 1}$. Also note that the sigmoid function is an element-wise operation therefore can be written in a fraction form.

And the cost (loss) function is defined as,

$$L(\mathbf{W}, \mathbf{b}) = -\frac{1}{m}\left(\mathbf{Y}^\intercal \log \hat{\mathbf{Y}} - (1 - \mathbf{Y}^\intercal) \log\left(1 - \hat{\mathbf{Y}}\right)\right) + \lambda \mathbf{W}^\intercal \mathbf{W} \tag{6}$$

where λ is *regularization parameter* to avoid overfitting. A larger λ makes the loss function cares more about not overfitting, while a smaller one decreases more loss.

B. Backward Propagation

Backward propagation performs a partial derivation for the aim of updating parameters of interests.

$$\mathbf{W} = \mathbf{W} - \left[\frac{1}{m}\mathbf{X}\left(\hat{\mathbf{Y}} - \mathbf{Y}\right) + \lambda \mathbf{W}\right] \tag{7}$$

$$\mathbf{b} = \mathbf{b} - \left[\frac{1}{m}\mathbf{X}^\intercal\left(\hat{\mathbf{Y}} - \mathbf{Y}\right)\right] \tag{8}$$

C. Optimization

In the optimization step, we use SAGA optimizer. Say we have an optimization function $f(x)$, and we would like to minimize the function, $f(x) = 1/n \sum_{i=1}^{n} f_i(x)$. As a contrast, the gradient descent algorithm try to do the following updates for each iteration k, given a learning rate, α, and suppose x is the parameter we try to learn,

$$x^{k+1} = x^k - \alpha \frac{\partial f(x^k)}{\partial x^k} \tag{9}$$

The SAGA algorithm, trying to keep a table of derivatives $\partial f_j(x^k)/\partial x^k$, with m entries (training samples), performs the following updates, for an arbitrary table entry j,

$$x^{k+1} = x^k - \alpha\left[\frac{\partial f_j(x^{k+1})}{\partial x^{k+1}} - \frac{\partial f_j(x^k)}{\partial x^k} + \frac{1}{m}\sum_{i=1}^{m}\frac{\partial f_i(x^k)}{\partial x^k}\right] \tag{10}$$

Besides SAGA optimizer, there are some others availabel, like newton-cg, lbfgs, liblinear and sag.

- The liblinear uses a coordinate descent (CD) algorithm. However, the CD algorithm implemented in liblinear cannot learn a true multinomial (multiclass) model; instead, the optimization problem is decomposed in a one-vs-rest fashion so separate binary classifiers are trained for all classes. This happens under the hood, so logistic regression instances using this solver behave as multiclass classifiers.
- The sag solver uses stochastic average gradient descent. It is faster than other solvers for large datasets, when both the number of samples and the number of features are large.
- The saga solver is a variant of sag" which we applied before.
- The lbfgs is an optimization algorithm that approximates the BroydenFletcherGoldfarbShanno algorithm, which belongs to quasi-Newton methods. The lbfgs solver is recommended for use for small data-sets but for larger datasets its performance suffers.

D. Results

We submit our predictions to official testing dataset, and achieved 97.48% area under the RoC curve.

VII. Statistical machine learning methods – support vector machine (SVM)

Similar to Logistic Regression we introduced above. In SVM, the basic idea is to apply a linear transform of input data, and add non-linear function. Then from loss function defined, optimize weight and bias parameters to find the best model.

Moreover, SVM is a maximum margin classifier, and with kernel it can solve nonlinear classification problem. This is much better than logistic regression.

A. Kernel Function

Kernel function is kind of function that can map input to a higher dimension output. But it is different from normal mapping, which means it gets the inner product of two vector in new space.

1) polynomial kernel:

$$k(\mathbf{x}, \mathbf{l}) = (\mathbf{x}^\intercal \mathbf{l} + \mathbf{a})^{\mathbf{b}},$$

where a is a constant and b is degree.

2) gaussian kernel:

$$k(\mathbf{x}, \mathbf{l}) = \mathbf{exp}(-\frac{\|\mathbf{x} - \mathbf{l}\|^{\mathbf{2}}}{\mathbf{2}\sigma^{\mathbf{2}}}),$$

where σ represent the variance of output.

3) linear kernel:

$$k(\mathbf{x}, \mathbf{l}) = \langle \mathbf{x}, \mathbf{l} \rangle .$$

B. Forward Propagation

Like linear regression, forward propagation can be represented by

$$\hat{\mathbf{Y}} = g(\mathbf{X}\mathbf{W} + \mathbf{b}) \tag{11}$$

And the cost (loss) function is a little different defined as,

$$L(\mathbf{W}, \mathbf{b}) = \frac{1}{2}\frac{1}{\|\mathbf{W}\|_2}, \text{ subject to } Y^{(i)}\hat{Y}^{(i)} \geq 1,\ i = 1, \ldots, n. \tag{12}$$

C. Optimization

Here we apply Lagrange duality to solve the problem. We define Lagrange function by multiplying a Lagrange multiplier α_i to constraints,

$$\mathcal{L}(\mathbf{W}, b, \alpha) = \frac{1}{2}\frac{1}{\|\mathbf{W}\|_2} - \alpha^\intercal(\mathbf{Y} \cdot \hat{\mathbf{Y}} - \mathbf{1}) \tag{13}$$

Then, the problem becomes

$$\min_{\mathbf{W},\mathbf{b}} \max_{\alpha_i \geq 0} \mathcal{L}(\mathbf{W}, b, \alpha) \tag{14}$$

While implementation, `sklearn` package help us to finish optimization process.

D. Result

We submit our predictions to official testing dataset, and achieved 71.28% area under the RoC curve.

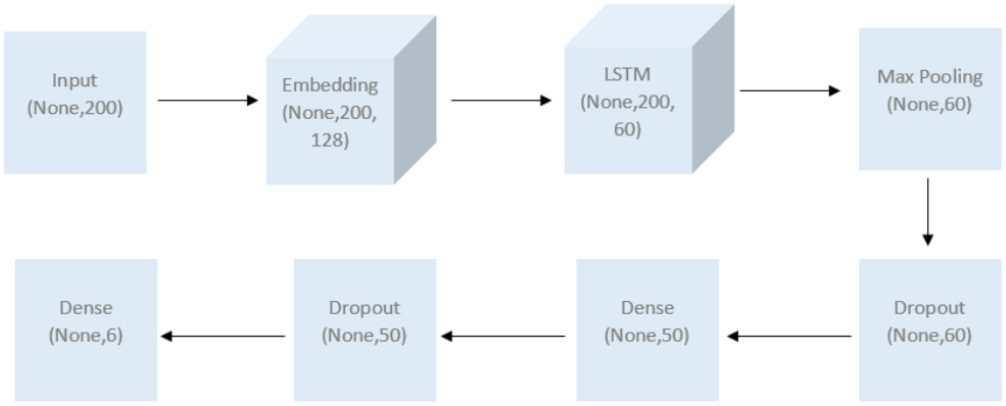


Fig. 6: Deep learning algorithm structure.

VIII. Data preprocessing for deep-learning

Before feeding raw comment text into the Deep Neural Network (DNN), it is almost certainly required that those comments are pre-processed – from which the DNN cannot understand if not processed – not like humans. The following steps summarize briefly the steps in our pre-processing procedure.

A. Word Tokenization

The main purpose of tokenizing comment texts lies in our model is word-based rather than sentence based, i.e., we perceive individual words as source of information though they come from sentences and contexts. Tokenization means break down sentences into words, as literal meanings.

B. Word Padding

After the tokenization process, words are transferred to vectors. However, they are not in the same length thus cannot be manipulated by our model. The aim of padding is to stuffing spaces (zeros in word embedding step, illustrated below) such that each vector shares pre-defined length. The maximum length is determined by empirical observations, as one can see in Fig. 2.

IX. Deep-learning enabled method

A. Structure of DNN

In Fig. 6, we picture the structure of our deep learning based algorithm. Note that the input is raw comment text, and the output is a vector (per input sample) consists 6 different numbers ranging from 0 to 1, representing the predicted probability regarding their categories, namely, `toxic, severe toxic, obscene, threat, insult, identity hate`.

B. Introduction to different layers

1) Embedding: Embedding layer represents word relationships with numbers. A very illustrative example could be, `Queen = King - man + woman`. As shown in Fig. 7, the words are clustered (in same color) using K-means method – if two words are closer (Euclidean distance) than the others, they are more likely appears in same context, and they share more presumably the same meanings. Note that there is an limit of extracting *features* out of one word, i.e., how many floats will be used to describe one such word. Apparently the more the better, yet more computational resources (especially memory) will be needed. In this project, we train our own embeddings – we would like to remain unbiased on the source of English corpus – which is more precise on particular problem. Note that the word embedding part contains the largest number of parameters that are needed to train.

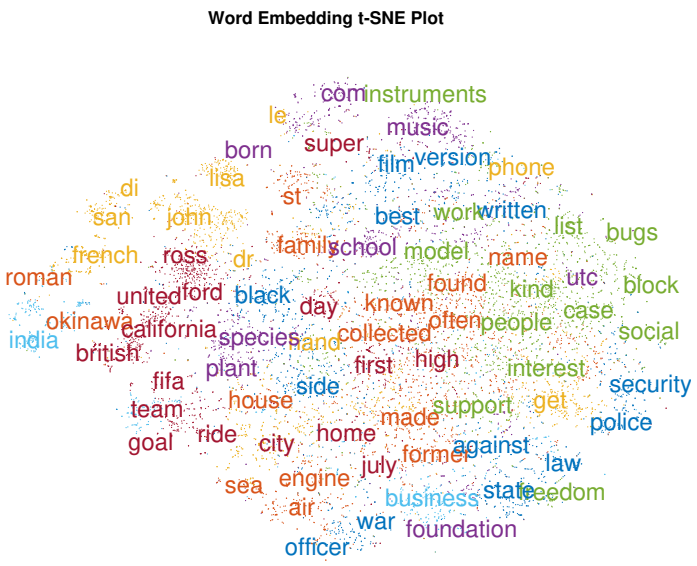


Fig. 7: Word embedding illustration

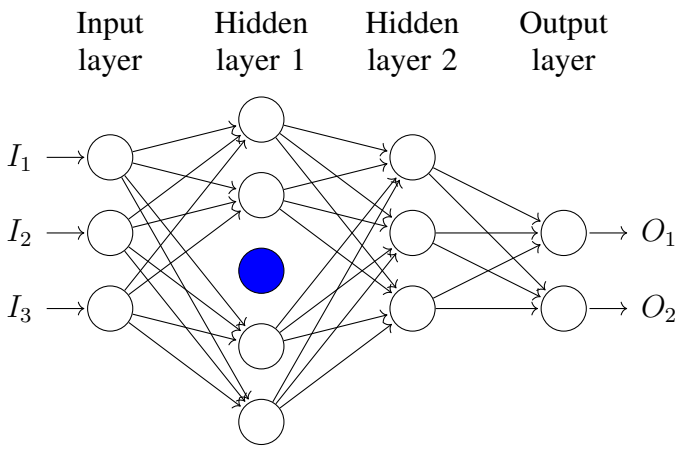


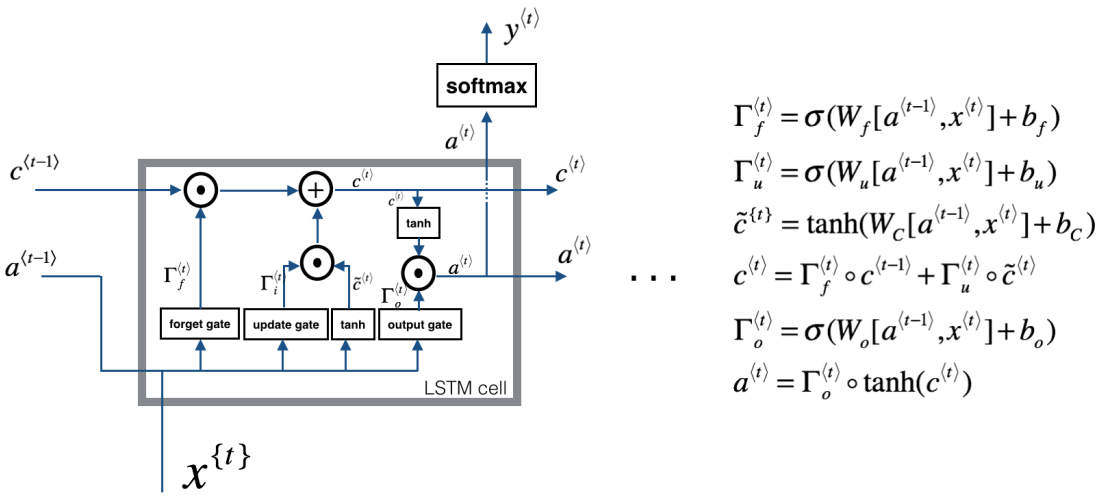
Fig. 8: Dropout, neuron shaded in blue is temporally turned-off during training

2) Dropouts: Dropout layer basically *turns-off* randomly some neurons in neural networks, as shadowed in blue of Fig. 8. Applying dropout layer can prevent overfitting and thus provide a way of approximately combining exponentially many different neural network architectures efficiently. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. Note that the dropout shall not be applied in testing (or validating) phase, i.e., no neuron shall be turned off.

3) Batch Normalizations: Batch Normalization layer can normalize the activations of the previous layer at each batch, i.e. apply a transformation that maintains the mean μ activation close to 0 and the activation standard deviation σ close to 1. The μ and σ are tunable hyper-parameters. Essentially, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

4) Dense layer: The dense layer is simply a routine fully connected neural networks, where each neuron is fully connect from the input to the output.

5) (Bi-directional) LSTM: Long Short Term Memory network (LSTM) is a special kind of Recursive Neural Networks (RNN) performs well on data with long-term dependencies. It was introduced by Hochreiter in 1997, and its structure is shown in Fig. 9. The term bi-directional, it does simply following



$$\Gamma_f^{\langle t\rangle} = \sigma(W_f[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_f)$$
$$\Gamma_u^{\langle t\rangle} = \sigma(W_u[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_u)$$
$$\tilde{c}^{\{t\}} = \tanh(W_C[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_C)$$
$$c^{\langle t\rangle} = \Gamma_f^{\langle t\rangle} \circ c^{\langle t-1\rangle} + \Gamma_u^{\langle t\rangle} \circ \tilde{c}^{\langle t\rangle}$$
$$\Gamma_o^{\langle t\rangle} = \sigma(W_o[a^{\langle t-1\rangle}, x^{\langle t\rangle}] + b_o)$$
$$a^{\langle t\rangle} = \Gamma_o^{\langle t\rangle} \circ \tanh(c^{\langle t\rangle})$$

Fig. 9: LSTM structure

– the first input sequence is its original sequence and the second one is a reversed copy of the input sequence.

- Forget gate is given by

$$\mathbf{\Gamma_f} = \sigma(\mathbf{W_f}[\mathbf{a}^{<t-1>}, \mathbf{x}^{<t>}] + \mathbf{b_f}). \tag{15}$$

This gate takes in $\mathbf{a}^{t-1}$ (previous state) and $\mathbf{x}^{<t>}$ (current input), and outputs a number between 0 and 1 for each number(information) in the cell state $\mathbf{\Gamma_f}$. 1 represents *completely keep this* while the 0 represents *completely get rid of this*.

- Update gate is given by

$$\mathbf{\Gamma_i} = \sigma(\mathbf{W_i}[\mathbf{a}^{<t-1>}, \mathbf{x}^{<t>}] + \mathbf{b_i}), \tag{16}$$
$$\tilde{\mathbf{c}}^{<t>} = \tanh(\mathbf{W_c}[\mathbf{a}^{<t-1>}, \mathbf{x}^{<t>}] + \mathbf{b_c}). \tag{17}$$

The forget gate and the update gate help update the cell state, $\mathbf{c}^{<t>} = \mathbf{\Gamma_i} \cdot \tilde{\mathbf{c}}^{<t>} + \mathbf{\Gamma_f} \cdot \mathbf{c}^{<t-1>}$

- Output gate is given by

$$\mathbf{\Gamma_o} = \sigma(\mathbf{W_o}[\mathbf{a}^{<t-1>}, \mathbf{x}^{<t>}] + \mathbf{b_o}), \tag{18}$$
$$\mathbf{a}^{<t>} = \mathbf{\Gamma_o} \cdot \tanh(\mathbf{c}^{<t>}). \tag{19}$$

C. Model compilations

1) Loss functions: A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model. In our project, a multi-class, multi-label problem, we use `binary_crossentropy` loss function and the last layer activation function is sigmoid. This loss function is mainly used to calculate the sigmoid cross-entropy between the predicted value `y_pred` and the true value `y_true`. It is mainly used in multi-classification tasks, which are not mutually exclusive. Also this function can output multiple labels with one input. In our project, we treat each category independently, which results us to use this loss function. Re-writing,

$$J(\mathbf{W}, \mathbf{b}) = -\frac{1}{m}(\mathbf{Y}^\intercal \log \hat{\mathbf{Y}} - (1 - \mathbf{Y})^\intercal \log (1 - \hat{\mathbf{Y}})) + \lambda \mathbf{W}^\intercal \mathbf{W} \tag{20}$$

2) Optimizer: The other required parameters for compiling a Keras model is optimizer. In this project, we select *Nadam* as an optimizer – *Nadam* has a stronger constraint on the learning rate and has a more direct influence on the updating of gradients. In general, *Nadam* is a better choice when *RMSprop* with momentum or *Adam* is supposed to use.

3) Metrices: A metric is a function that is used to judge the performance of our model. `Metric` functions are to be supplied in the metrics parameter when a model is compiled. A *metric* function is similar to a *loss* function, except that the results from evaluating a metric are not used when training the model. Moreover, a metric can measure our model by some non-differentiable function such as accuracy. In our model, we display accuracy, and Mean Squared Error (MSE) as our metrices.

D. Results

The deep learning enabled methods takes 48 hours to train on a quad-core computer (embedding is to large for GPUs), and the best validation accuracy (on a self-divided training-validation-test dataset) is 98.18%, and the submission shows this method achieved, 97.37%.

X. Source code

All the source code is available at GitHub, https://github.com/DuanRuixiao/Projects.