Introduction

Discussing things, you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. Thus, building a model to detect the negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion) becomes meaningful.

The purpose of our project is to build a model to detect toxicity across a di- verse range of conversations, which recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities. Thus, we have built an NLP model with DNN that recognizes toxicity and minimizes this type of unintended bias with respect to mentions of identities. The dataset would be labeled for identity mentions and optimizing a metric designed to measure unintended bias.

Our model will be separated to two part:

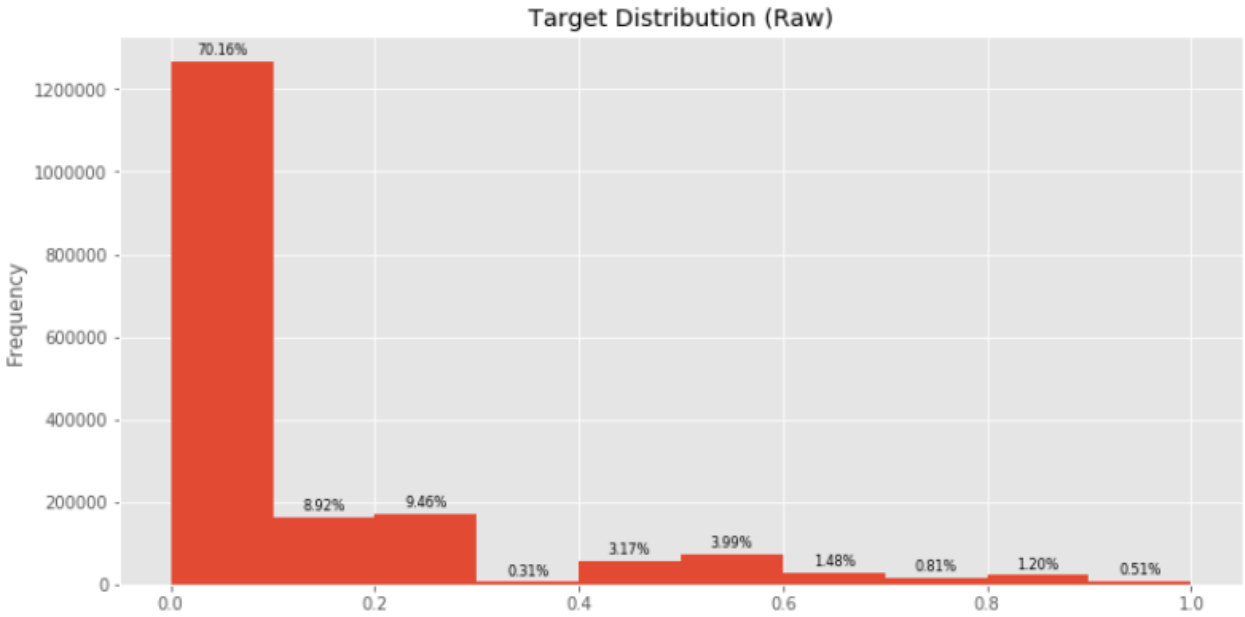
1) Data pre-processing. In this part, we will clean the data by delete some symbols like: “ /-'?! π‘₹´”. And we will also do the word embedding by converting the words to vectors. For the word embedding part, we will use the pre-trained word vectors, GloVe which is trained by Jeffery Pennington, Richard Socher and Christopher D.Ming and FastText which is released by Facebook. Then we concatenate them to get a new word vector.

2) DNN. For the deep learning model part, we use two BLSTM as the main sequential training model and then we use two kinds of Global pooling layer to extract feature points. After pooling, we use the Res-Net to do the full connect nonlinear layer. At last, the output layer will use a sigmoid activation function

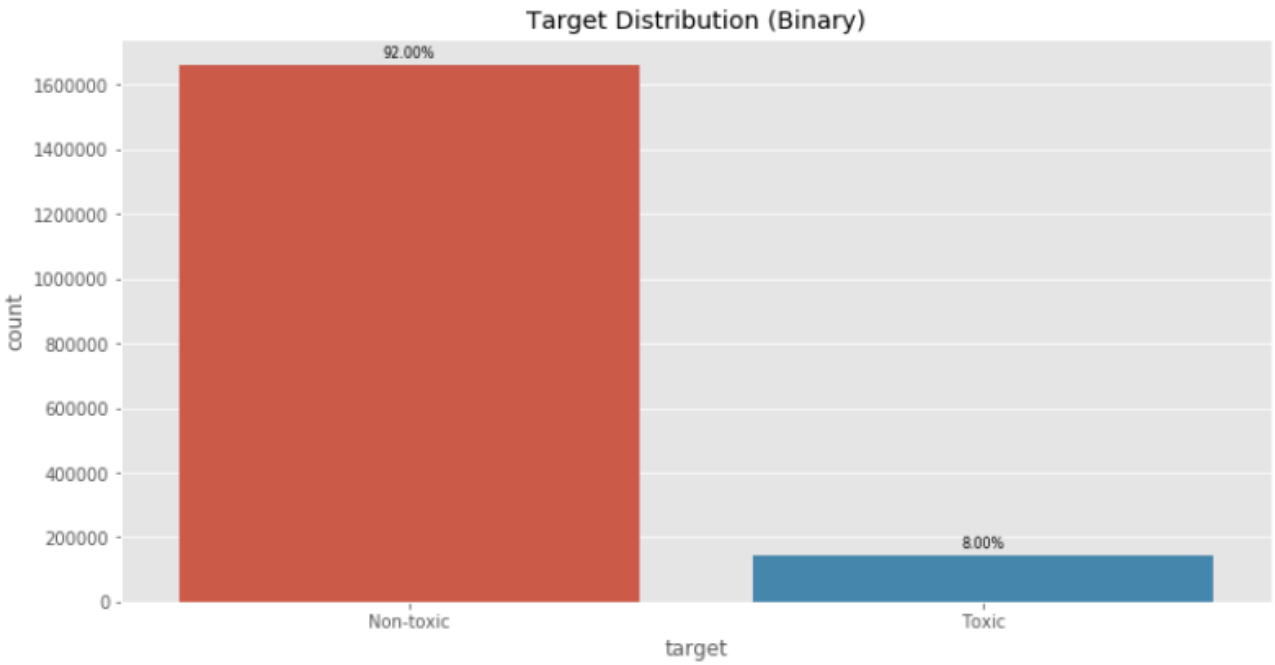
Description of Dataset

We have 1804874 entry in our training set, and 97320 entry for our test set. As the most of the Kaggle, we will use part of test set to calculate our public leaderboard.

The label in our training dataset is given by the continues value in `[0,1]`, the raw distribution will be

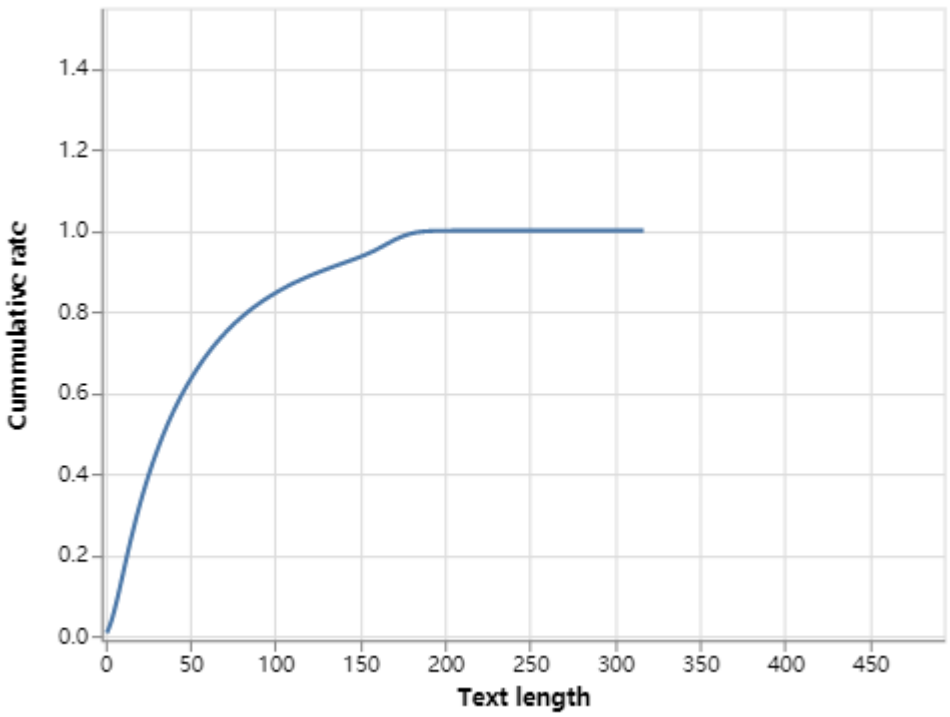


if we choose threshold of `0.5`, the distribution will be



Clearly our distribution is high **unbalance**

The length of our comment text is distribution as following, most of our comment has less than `200` words, and the maximum words in the sentence is within `220`.



Above is the distribution of number of words in comments.

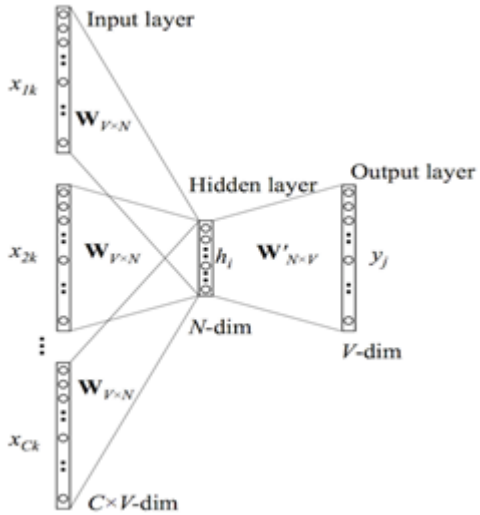
Description of the deep learning network and training algorithm

Word Embedding

Word2vec

Word2vec is a group of related models that are used to produce word embed- dings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

Continuous bag-of-words (CBOW) is a typical architecture of Word2vec. In CBOW, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction.



In the continuous bag of words model, context is represented by multiple words for a given target words. For example, we could use “cat” and “tree” as context words for “climbed” as the target word. This calls for a modification to the neural network architecture. The modification, shown below, consists of replicating the input to hidden layer connections C times, the number of context words, and adding a divide by C operation in the hidden layer neurons.

Our pretrained word vectors FastText is trained by this method.

GloVe

Global vectors for word representation using a combination of the Matrix and window size methods. The main method is that we set a window size, and based on the size we calculate the frequency of word occurs with word j .We use $X_i j$ represent the frequency that word i and j are in the same window. Our purpose is to train a NN model to get a vector which has a close relationship with the frequency of X_{ij}

$W_i^T W_k + b_i + b_k = log(X_{ik})$

And we can find that w_i and w_k are symmetric vector. And b_i and b_k are bias for w_i and w_k. Thus, for the whole model, our target function is

$J = \sum_{i,j=1}^{V} f(x_{ij})(W_i^T W_k + b_i + b_k - log(X_{ik}))$

where *V* is the size of the vocabulary. The weight- ing function should obey the following properties:

1. *f* (0) = 0. If *f* is viewed as a continuous function, it should vanish as *x* → 0 fast enough that the limx→0 *f* (*x*) log2 *x* is finite.
2. *f* (*x*) should be non-decreasing so that rare co-occurrences are not overweighed.
3. *f* (*x*) should be relatively small for large values of *x*, so that frequent cooccurrences are not overweighed.

BRNN and BLSTM

In this project, I first invest the possible of applying Bidirectional Recurrent Model into our problem. I came cross this model in one of the kernels. I basically solve the following problems:

- Why do we need Bidirectional model?
- Does it improve our score?

- How does it works in frameworks, what params do we have?
- Is there downside for Bidirectional model?

Starts with first question. Let's take a step a back and think why we RNN works for NLP problem. One of the reason is that RNN can "remember" the information from previous words, which is the same like how human read. But this method has its own method, one example of this problem is, suppose we have a classifier to determine every word in sentence is a name or not. Given those two sentence

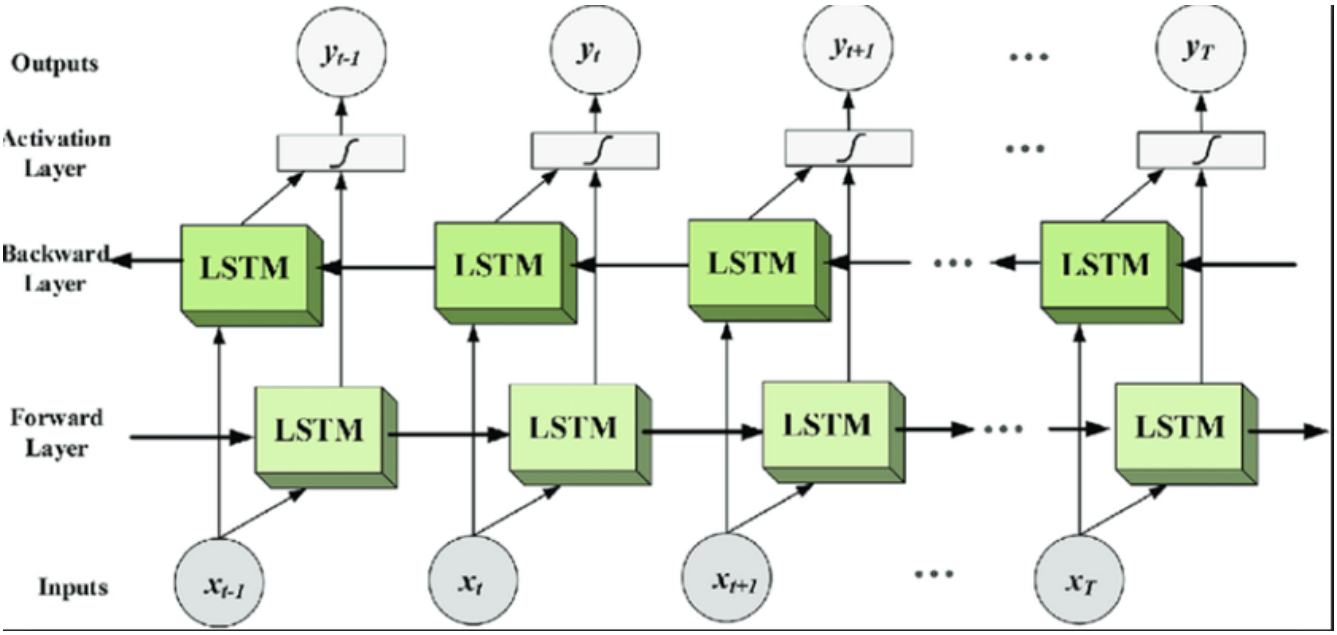
> He said, "Teddy bears are on sale!"
>
> He said, "Teddy Roosevelt was a great President"

It would be hard for an RNN to do the job, because the information for those two sentence before `Teddy` are the same. Thus, we need somehow "capture" the information after `Teddy`, That's what Bidirectional RNN can help us.

But how BRNN "capture" the information? BRNN has two RNN cells inside. One will operate on the first word of sentence, and move to the second one, ... etc. The other will operate on the last word of sentence, and move to the previous one, ... etc.

In Keras, there is a layer wrapper call Bidirectional to make a unidirectional layer to become bidirectional. It accepts the Recurrent layer(RNN, LSTM, GRU) instance as input. And there is another param to determine how will we merge the result of the two Recurrent layer. I did test over difference merge_mode and its impact over our model performance.

But to using BRNN, one thing we need is that we need whole sequence as input for the two RNN cell to read all the input before we merge them in some way and get the result.



The result of BLSTM can be expressed in such way

$$\overrightarrow{h_t} = H(W_{x\vec{h}}x^t + W_{\vec{h}\vec{h}}\vec{h}_{t+1} + b_{\vec{h}})$$

$$\overleftarrow{h_t} = H(W_{x\overleftarrow{h}}x^t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}})$$

$$y_t = W_{\vec{h}y}\vec{h}_t \oplus W_{\overleftarrow{h}y}\overleftarrow{h_t}$$

Note that the way for us to combine the result of two LSTM cell can be different. Here we just concatenate the two vector. But we can also sum them up or do an average.

The downside of this model compared with LSTM or other RNN cells can be very simple. First, as we employ two LSTM cells in one RNN cells, the training time of the BLSTM cell should be doubled for the same situation. Also, for the LSTM cells in BLSTM to work, the whole sequence must be input, which means you can not use a word as a input in our problem, the input can only be the whole sentence.

LearningRateScheduler

LearningRateScheduler is a callback provided by keras, which main focus on adjust learning rate according to the epoch index and current learning rate. Here we use $0.001 * 0.6^{\text{epoch}}$ as our learning rate, as we can expect that the smaller learning rate will help us converge in the later epoch.

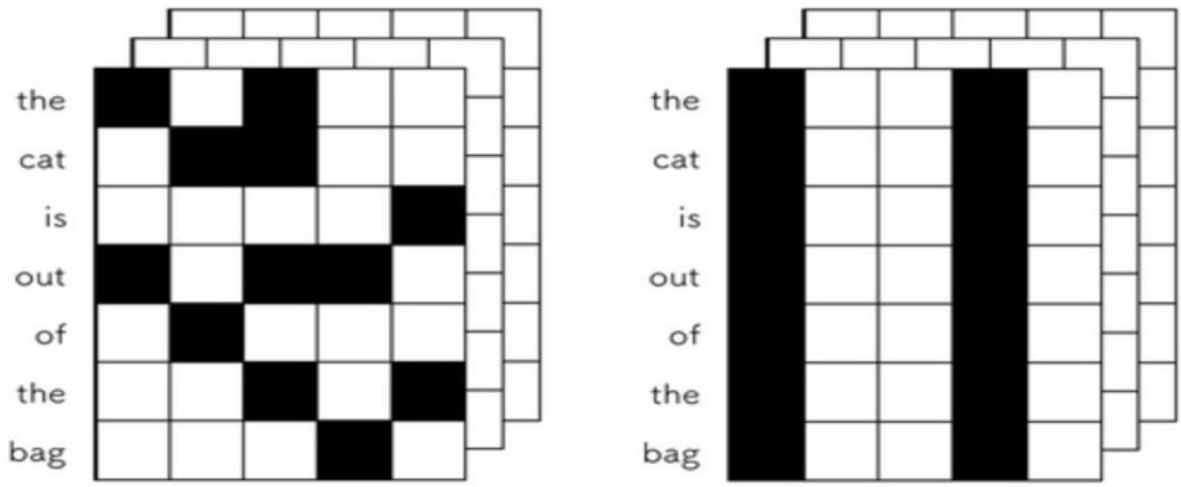
Pooling layer

We use pooling layer in our model for two reasons.

- Since the output of the BLSTM is twice of LSTM if we did not specific merge_mode, we can expect the output of the BLSTM contains lots of duplicate information, using `GlobalMaxPooling1D` and `GlobalAvgPooling1D` will help us reduce the size of tensor.
- It also helps fighting again overfitting, since it reduce the complexity of the network.

Spatial Dropout

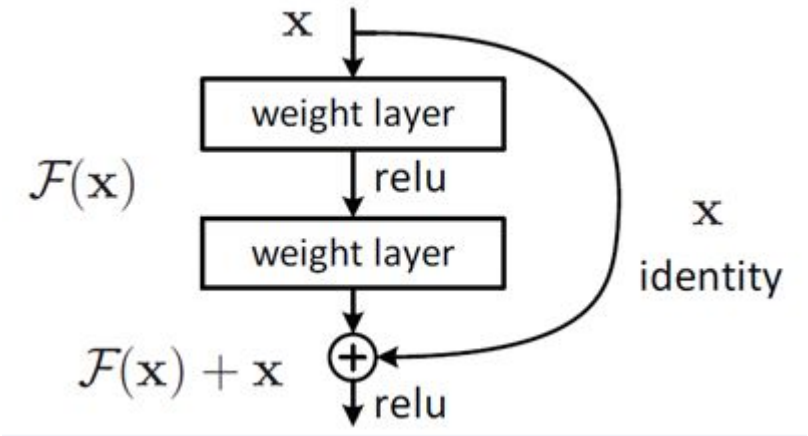
I use the spatial dropout instead of normal dropout to avoid overfitting. For normal dropout, we randomly select elements in tensor and set them to be zero. But for the spatial dropout, we will randomly select dimension and set all elements in that dimension to be zero.



The picture shows the difference between dropout and spatial dropout. As we can see that the left picture is the normal dropout, and the right part is spatial dropout.

Residual Network

Let us consider F(x) as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers. If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions2, then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., F(x) + x (the input and output are of the same dimensions). So rather than expect stacked layers to approximate F(x), we explicitly let these layers approximate a residual function H(x) = F(x) + x. The benefit of Res-Net is that we can increase the complexity of the model without increasing too many computations. And by increasing the complexity, we can get better result.



Experimental setup

We use the train dataset and test dataset as Kaggle gives us.

We will use Keras with tensorflow as backend, which will provide ability to plot via tensorboard.

Yes, we will use minibatches.

We will determine the size with experiment.

For learning rate, we use a callback to change the learning rate based on our index of epoch.

We will add spatial dropout layer to reduce the overfitting.

Result

1. merge_mode of BLSTM

Experiment has shown that we will get public score as `0.92836` if we sum the vector up and `0.93061` if we choose to concatenate them.

2. Number of layers of BLSTM

I also test over the influence of different layer of BLSTM, when we have 2 layer BLSTM, we have public score of `0.93010`, when we have 3 layer BLSTM, we have `0.92960`, but for 4 layer BLSTM, we have public score `0.93043`.

3. BLSTM v.s. LSTM

I also test over how BLSTM's performance differs from the LSTM's performance. In our test, we have public score of `0.93010` if we use 2 layers of BLSTM, but we will have public score of `0.92821`. However, the running time of the BLSTM version is `6983` s while LSTM version only costs us `3267` s.

4. BRNN cells choice

If we choose to use BGRU, the public score will be `0.92943`, while BLSTM could get us `0.93010`

5. Choice of Word embedding library

If we use FastText alone, our public score will be `0.92870`, if we use GloVe alone, our public score will be `0.92783`

If we concatenate GloVe and FastText, our public score will be `0.93010`

6. Choice of optimizer

If we choose Adam as optimizer, then we will get public score of `0.93010`, if we choose AdaDelta as optimizer, then we will get public score of `0.90783`

7. Size of mini-batch

If we choose batch size of 1024, we will get public score of `0.92825`, if we choose batch size of 256, we will get public score of `0.93039`, if we choose batch size of 512, we will get public score of `0.93010`

8. Learning rate

We test over some value of the decreasing factor, when we use `0.6`, we have public score `0.93010`, but when we have `0.9`, we have public score `0.92969`

9. With Spatial drop or not

With the spatial dropout, we have public score of `0.93010`, while if we do not have spatial dropout, we have public score of `0.93095`

10. Output of LSTM unit

If we have 128 as the output shape of LSTM unit(which means 256 for BLSTM layer), we have public score of `0.93010`

If we have 64 as the output shape of LSTM unit(which means 128 for BLSTM layer), we have public score of `0.92308`

Clearly, the less of LSTM unit output would carry less info, thus, decreasing the performance.

Summary and conclusions

In this project, we test over some of the classical NN structure in NLP problems like LSTM, BLSTM, GRU, BGRU. We have learn, understand and experiment over the following topic:

- Relationship between Recurrent model and NLP
- Bidirectional RNN and it variant.
- Word Embedding algorithm
    - Word2vec

    - Glove
- Residual Neural Network
- Global Pooling and its application in NLP

For future improvement, we have following ideas in mind.

- Using BERT embedding, as it is the latest breakthrough made by Google recently.
- Using Attention layer to reduce the bias in the model.
- Using deeper model and better hyperparameters.
- Deal with unbalance data.

Reference

1. Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).
2. Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.
3. Yu, Z., Ramanarayanan, V., Suendermann-Oeft, D., Wang, X., Zechner, K., Chen, L., ... & Qian, Y. (2015, December). Using bidirectional lstm recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*(pp. 338-345). IEEE.
4. https://www.kaggle.com/bminixhofer/simple-lstm-pytorch-version

Appendix

```
import numpy as np
import pandas as pd
from keras.models import Model
from keras.layers import Input, Dense, Embedding, SpatialDropout1D, Dropout, add,
concatenate
from keras.layers import CuDNNLSTM, Bidirectional, GlobalMaxPooling1D,
GlobalAveragePooling1D
from keras.preprocessing import text, sequence
from keras.callbacks import LearningRateScheduler


EMBEDDING_FILES = [
    '../crawl-300d-2M.vec',
    '../glove.840B.300d.txt'
]
NUM_MODELS = 2
BATCH_SIZE = 512
LSTM_UNITS = 128
DENSE_HIDDEN_UNITS = 4 * LSTM_UNITS
EPOCHS = 4
MAX_LEN = 220
```

```
def get_coefs(word, *arr):
    return word, np.asarray(arr, dtype='float32')


def load_embeddings(path):
    with open(path) as f:
        return dict(get_coefs(*line.strip().split(' ')) for line in f)


def build_matrix(word_index, path):
    embedding_index = load_embeddings(path)
    embedding_matrix = np.zeros((len(word_index) + 1, 300))
    for word, i in word_index.items():
        try:
            embedding_matrix[i] = embedding_index[word]
        except KeyError:
            pass
    return embedding_matrix


def build_model(embedding_matrix, num_aux_targets):
    words = Input(shape=(MAX_LEN,))
    x = Embedding(*embedding_matrix.shape, weights=[embedding_matrix], trainable=False)
(words)
    x = SpatialDropout1D(0.3)(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)
    x = Bidirectional(CuDNNLSTM(LSTM_UNITS, return_sequences=True))(x)

    hidden = concatenate([
        GlobalMaxPooling1D()(x),
        GlobalAveragePooling1D()(x),
    ])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    hidden = add([hidden, Dense(DENSE_HIDDEN_UNITS, activation='relu')(hidden)])
    result = Dense(1, activation='sigmoid')(hidden)
    aux_result = Dense(num_aux_targets, activation='sigmoid')(hidden)

    model = Model(inputs=words, outputs=[result, aux_result])
    model.compile(loss='binary_crossentropy', optimizer='adam')

    return model


def preprocess(data):
    '''
    Credit goes to https://www.kaggle.com/gpreda/jigsaw-fast-compact-solution
    '''
    punct = "/-'?!.,#$%\'()*+-/:;<=>@[\\]^_`{|}~`" + '""“”’' + '∞θ÷α•à−β∅³π‘₹´°£€\×™√²—–&'
    def clean_special_chars(text, punct):
        for p in punct:
            text = text.replace(p, ' ')
        return text
```

```
    data = data.astype(str).apply(lambda x: clean_special_chars(x, punct))
    return data


train = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/train.csv')
test = pd.read_csv('../input/jigsaw-unintended-bias-in-toxicity-classification/test.csv')

x_train = preprocess(train['comment_text'])
y_train = np.where(train['target'] >= 0.5, 1, 0)
y_aux_train = train[['target', 'severe_toxicity', 'obscene', 'identity_attack', 'insult',
'threat']]
x_test = preprocess(test['comment_text'])

tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(list(x_train) + list(x_test))

x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
x_train = sequence.pad_sequences(x_train, maxlen=MAX_LEN)
x_test = sequence.pad_sequences(x_test, maxlen=MAX_LEN)

embedding_matrix = np.concatenate(
    [build_matrix(tokenizer.word_index, f) for f in EMBEDDING_FILES], axis=-1)

checkpoint_predictions = []
weights = []

for model_idx in range(NUM_MODELS):
    model = build_model(embedding_matrix, y_aux_train.shape[-1])
    for global_epoch in range(EPOCHS):
        model.fit(
            x_train,
            [y_train, y_aux_train],
            batch_size=BATCH_SIZE,
            epochs=1,
            verbose=2,
            callbacks=[
                LearningRateScheduler(lambda epoch: 1e-3 * (0.6 ** global_epoch))
            ]
        )
        checkpoint_predictions.append(model.predict(x_test, batch_size=2048)[0].flatten())
        weights.append(2 ** global_epoch)

predictions = np.average(checkpoint_predictions, weights=weights, axis=0)

submission = pd.DataFrame.from_dict({
    'id': test['id'],
    'prediction': predictions
})

submission.to_csv('submission.csv', index=False)
```