Toxic Comment Classification Challenge competition writing report (top 5% , 184/4551, solution and experiments)

Data set:https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

Chan Yao Chun/詹耀鈞

Disclaimer: the content might have some words or sentences be considered profane, vulgar, or offensive. and all of them are the competition's data content.

- **Abstract:**

This data is the comments from Wikipedia's talk page edits, and the goal is to detect which comment belongs to which kind of offensive action. The output is probability, which means the model thinks how many probability the comment will belong to the kinds of abuse, and there are six type of abuse, toxic, severe_toxic, obscene, threat, insult, and identity hate.At final, the competitor will be ask to build the multi head model to make the prediction.

About the model, the holder of the competition has mentioned the paper: "Attention is All you need [https://arxiv.org/pdf/1706.03762.pdf]" , and I try other different type of attention layer in this competition. Moreover, I'll discuss other inference of different experiment in this competition.

About toxic comment research effort was started by **Jigsaw** and the **Google Counter-Abuse Technology Team (Alphabet Inc. is their parent). Also, the competition is hold by them, and it is part of the Conversation AI project.**

- **Data exploration:**

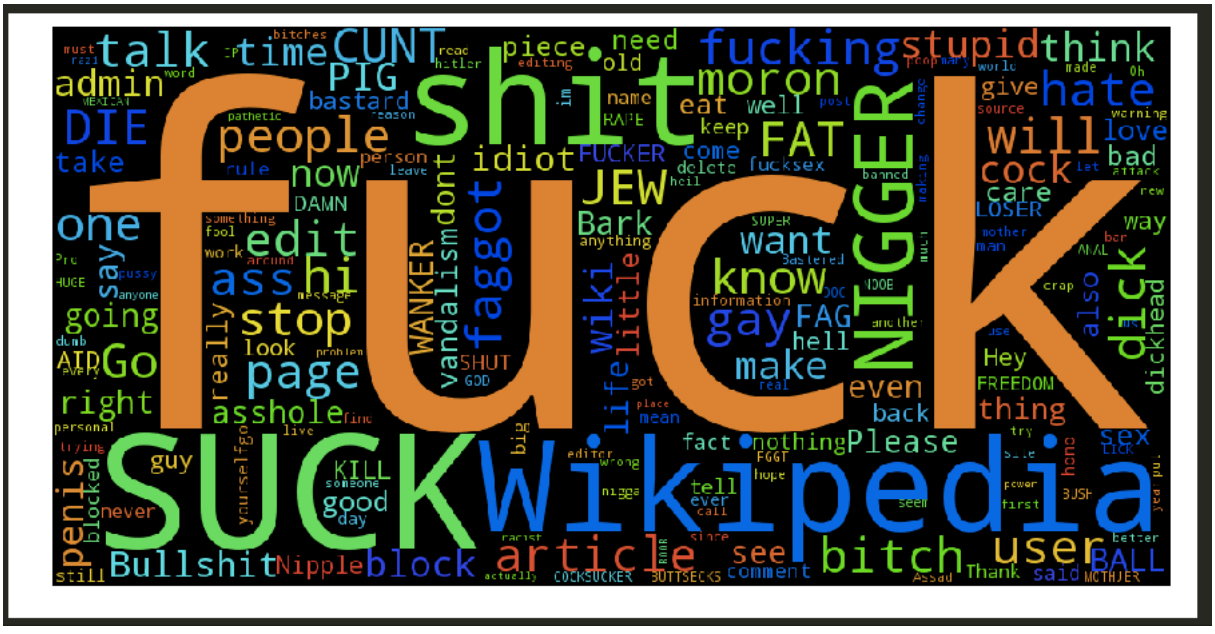

figure1: word frequency in toxic comment represents by word cloud

The data includes comments from the wiki, and it also include multi language comment, so it's hard to process, I have tried to use google translate API, but get suck in some case, so it needs to filter out the long word sequence and do the proper encode, decode task.

There are six type of multi label and is another difficult problem, the data set of them are unbalance, as **figure2** .Thus, the models are usually have bad result at the **threat** column. The worse thing is that the training and testing data is different distribution, if we use adversarial validation or t-SNE can observe these condition.**Note: 1

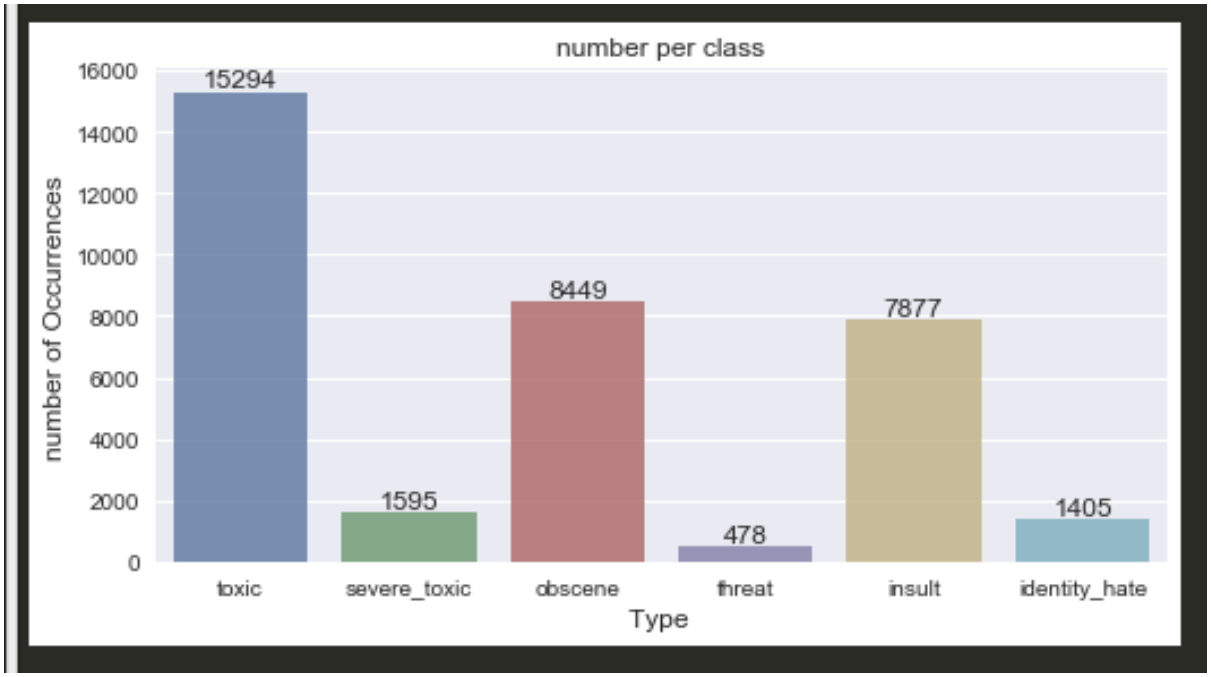

figure 2: The **toxic** data's multi label distribution, there are six type of labels, and the whole training set's data volume are ten times of them**(159571)**.

Note 1: adversarial validation is like https://www.kaggle.com/konradb/adversarial-validation (I take quick reference from this script), labeling the training data with 0, testing data with 1, and use the algorithm to detect if they are easy to identify the difference between training data and testing data.

- **Main part of feature engineering(NLP):**

For the correction method, I first use brute force to correct some words, base on my own knowledge, but the result it's not improve a lot. Later, I try to borrow the algorithm of Peter Norvig **[14]**, Director of Research at the google, however in this competition, it makes marginal improvement, because some of the tokens or swear word really important, and correction algorithm may miss correct them, For example, if the word "fucjjjk" will be correct as "fulk", but maybe the writer wanted to type in "fuck". Also, due to time I found it is close to the deadline of the competiton, so I do not use it at final, but according to other competitor, it may give a diversity to the model, and the model maybe more stable at the private score. After the competition, I slightly modify the Peter Norvig's algorithm using glove embedding as correction dictionary, it can be done in less than three hours, faster than 8 times compared to the original way of computation (On the 4 cores machine, and training other model at the same time). **Note 2.

On the other side, I found that tokenize some punctuation is really help the improvement. Even I transfer some of the punctuation to the words, like emoji or something, some punctuation still be ignored, so it is important to do tokenize these punctuation and let the model to justify it, especially there are many swear words are relate to some punctuation. Thus, I only remove - , /, __,…etc,

because for most of the case they are just concat the sentence, but others like !, ? , related to the sentence function or the emotion to be displayed, I kept them all.

Another processing I did is that I replace the meaningless duplicate character in a word, like "fucccccccck" transfer to "fuck", because the extra "c" characters do not have special meaning to me. Yet, in the character level embedding, it's better not to do it, I'll discuss later.

At the final step of preprocessing, I use word net lemmatization to reduce inflectional forms of word to let the model be more stable. The pros of lemmatization is that it can let the words become more clear to the model, that is, the similar meaning of the words will not be miss up, because the words are lemmatized to common base form by grammar(tag in nltk library). According to local cross validation, it shows that the models train with the lemmatized data are less to overfit at the process of training, also it's final score can get improve by 0.0002, it's marginal, but and a different to this competition. Yet, due to making diversity, I just let one of my final model to feed this kind of data.

The rest part of the preprocessing is getting out of some leaky information, like user ID, they are not the same in training set and testing set, and base my personal opinion, I think that the character and some special punctuation are more important information to the model.

One thing to mention is that I do not use data translate argument, but I will study the method in the future to refine its performance. ****Note 3.**

****Note 2:** if you want to get the faster correction, can use TextBlob library, it's also base on Peter Norvig's algorithm. Yet, it's can not use word embedding index to do the correction.

****Note 3:** the origin design is from one kaggle competitor, Pavel Ostyakov, the method dose not translate all the sentence directly to English, it uses "indirectly" way, or say encode the sentence to another non-English language and decode it back to English, in order to get the precise result, less overfitting, and easier for batch processing. For example, all sentence transfer to different non-English language, like German, France….,etc, and translate them to back to english, using these data as the argument for the training data, it will have extra **N*M** data size for training, **N** is the number of non-English language we use to translate, and the **M** is the original data size we have.

- ## Models' architectures:

In this text kind of competition, Deep learning neural network model is the mainly used model. Since it performance can not easily be beat by other kind of model, like SVM or logistic regression, though in the final stage of model ensemble, all of the different type models have same degree of importance, because they will provide diversity for us.

I also try a lot of different models' architectures, except the output layer I use six nodes of sigmoid function for building multi head model which solves six type of multi label problem. I'll discuss all the experiment relate to deep learning neural network model I done in this competition:

1. Embedding layer:

Using NN models to solve the text classification problem, one of the most important part is the embedding layer, it really determine model's performance really significantly, because it's like a feature engineering for NN model.

I use Fasttext and Glove(including Glove Twitter) as my pre-trained word embedding models **[1 , 2]**. These two embedding have nice performance in these competition, because of their

enough vocabulary, also they might be more similar to the words in the training and testing data, because most of them are came from web crawl or wikipedia.**Note 4.

combine with char embedding:

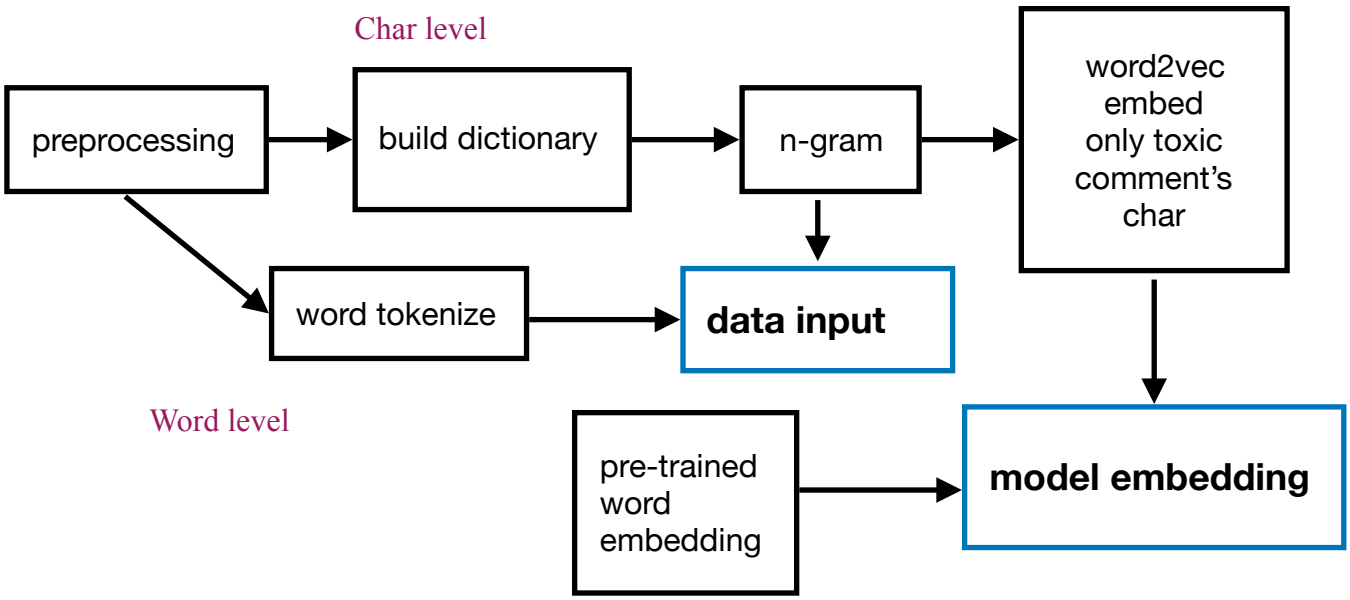

figure 3: flow chart represent embedding and the data input

For getting the detail of the word structure, especially the word in the toxic comments which usually contains a non-standard word or unusual sentence structure, so I use n-gram to let the word be repeat by char level in n times to impose model get the information of these non-standard word. After training this way, I put the char level n-gram sentence of toxic comment to Word2vec model get the word2vec char level embedding. Why use toxic comment to train Word2vec embedding only? Except the reason I mentioned above, also because in t-SNE analysis, the result shows that the training and testing data "overlap part" is almost at the toxic comment. Thus, to get the better result of private and public leaderboard score, I use this strategy.

Next step, I combine char level and the word level embedding together to feed my RNN model, and for Glove vector is improved most, however for Fasttext wiki is very marginal, even worse, I think it's because the Fastext model already trained with subword information, that is, it uses bag of character n-grams approach training method which is similar to my method. Also, sometimes, the long sequence of char level embedding will let RNN model lost memory easily.

The detail of implement is that I use multiple input and parallel learning of single GRU, because the char level embedding and word level embedding are not the same.

**Note 4: After the competition, I found that actually there are some non-English words that are included in wiki.en.bin, like "你好", "真", "幹"...etc. Most are short words.

Note 5: The reason that word2vec pe-trained model not use there may be the ski-gram and bow ignore the internal structure of the word, but the bag of character n-gram will get the information[**1] ,and it is important to the toxic comment, so the result come out of it.

On the other side, my method of character level to train word2vec is treat one n-gram characters as a word (or say basic element to train the model), however, the Fasttext wiki is still word level embedding, but each word has its own bag of n-gram character vectors.

2.RNN layers / After embedding layer:

I use bidirectional GRU/ LSTM for my RNN model, according to my previous experience and project, it has a better result compared just single recurrence direction RNN, because the bidirection provide the model with more stable performance and deeper memory recurrence, and in this data set , the above description is quit matched to bidirectional GRU/ LSTM's performance.

Moreover, I use skip connect **[7]**, to one of my bidirectional LSTM model, it gets low level and high level features, so the performance is better then general model. Yet, I did not use the features right after the embedding layer, because I just use 100 units of LSTM for preventing overfit, so in this case, the embedding layer's feature will dominate the information because it has much higher dimension than RNN layer.

A little detail is that I use cuDNN as a faster implement for RNN layers. Because of that, it is unavailable to use recurrent drop out, also, it's easy to get overfit in this data set, typically after 8 epochs will start to overfit, even with bidirectional RNN layers, so I put the Spatial drop out after the word embedding layer for stronger regularization(for each model). The best thing of this kind of drop out is that it will drop the high dependent features and also drop the "whole channel", so in these kind of text classification problem, the information after Spatial drop out will not confused the model compares to the original version of dropout, because using simple dropout will let feature map not be consistent, that is, some of features map will be seem in the front layers, but some are not seem in the last layers, so it is more like adding a noise to the NN model.

3.Three type of attention:

These different type of attention appear in three papers, and actually they are very similar. The first is the Attention mechanism in feed forward model for temporal data **[3]** (I'll call it feed forward attention), in this case, the attention mechanism does not specify to solve the text problem. Also, the attention layer in the paper does not consider temporal order, because it computing an average over time discards order information. Indeed, I put it at the last layer for my bidirectional RNN model to do some experiment, and get the good result in some case, like concat it with maximum and average value of each sequence, or feed to the fully connected neural network at the last stage, but it get worse result when I only feed it directly to the output layers without aggregate with other layers.

The second type of Attention mechanism is more specify on the document classification problem **[4]**, and computation is more complicated which consider the word similarity. I use this Attention(I'll call it Attention with context) layer in the Hierarchical Attention Networks, just as the paper, sadly, its performance is very unstable in the 10 fold cross validation, though it has mediate performance on the leaderboard but can not out beat other models in the final stage, for saving the computation power, I get rid of this chose. Yet, after the competition, I did a simple experiment and found that the second type of Attention mechanism get better performance than other two type Attention mechanism on recognizing the temporal order if the relative order of the data is not broken.**Note 5**

The third type of Attention mechanism**[7]** (I'll call it weight average attention)is very similar to the first type Attention mechanism, their formula are:

$a(h_t) = \tanh(W_{hc}\, h_t + b_{hc})$, $e_t = a(h_t)$, **(first type)**

$e_t = h_t W_a$, **(third type)** **Note 6

After compute e_t, they all go to normalize at softmax function, then get the weighted average, base on the performance, seems that first type Attention mechanism getting extra tanh will get the worse result in this competition. I think that because of previous layers are merely all tanh, so it have a little gradient vanish or overfitting. Also, the value which feed to the attention layer may not centered at 0, so it will be bad for single tanh to deal with it. Yet, the result also show that the temporal order at the final architecture of RNN model may not be very important, this also mentioned in **[3]**

**Note 5: the intuition is adding constant noise to the data without breaking the relative order of the sequence in the data and see whether the model can recognize it.

First, set input data X to be an 10X5 array of sequences that represents 1 to 10 by binary code, so there are 10 vectors and each represents different integer, and the Y is a 10X10 array of sequences that represent 1 to 10 by one hot encode, for example, 1 is [0,0,0,0,0,0,0,0,0,1], 5 is [0,0,0,0,0,1,0,0,0,0], 10 is [1,0,0,0,0,0,0,0,0,0]. After feed X, Y to train the models with different attention layers , I start to use three different data to predict:
(A). part of X but shuffle the order of vectors
(B). use the (A) data and add 1 to every elements
(C). use the (A) data and add 10 to every elements
(D). use the (A) data and add 100 to every elements
Compare to the other two Attention mechanism, the result shows that Attention with context can get the most similar performance in these four cases and outperform in case (B), (C), (D), that is, it is more good at recognizing the order of sequence even when it did not see the same data before, and it just depends on the relative order of elements in the sequence, like big or small.

In this experiment, I use three different seeds for initial weight of models to do three times experiment. That is to make sure I get the more concrete result, and embedding layer's dimension is 10 to 10, after embedding layer is only attention layer.

One thing which needs to be mentioned is that the attention layers performance are also affected by previous layers' parameter, like LSTM, and this is a simple experiment. If I have time in the future, I'll dig in the relative topics deeper.

**Note 6: the author of this paper put some trick to enhance numerical stability(In the code). Maybe because the small problem of tensorflow backend. Yet, this is not mentioned in paper.

4.Pooling to fully connected layer

For fully connected NN model to extract the best features, I concat the max and average value of each sequence, because it will not only focus on important information, that is, largest output value of each sequence.There is a key detail can influence the performance of model a lot, according to**[8]**put the batch normalize before any of drop out layer will let the inconsistency of variance less likely happened. Thus, my model's numerical behavior will get more stable. Base on my experience consequence, if using the idea mentioned in the paper, each fold's score will be more closer and better.

Also, I set activation of fully connected network to be the elu, so the output value will not miss the negative input, and get gradient vanished in this case, also its computation is not slow.In my experiment, it really get better performance than just using relu function.

One thing to mention is that I try just return the last input to the fully connected layer instead of using pooling layer, and the result not bad, though I did not use it at the final stage, maybe can concat it with pooling layer at the future.

5. CNN

CNN is the key to enhance the set of model's diversity. I do the experiment on Yoo Kim's CNN, but the result is bad, maybe because the model is not deeper enough.

On the other side, I use a deep CNN modified from Resnet, and get the better result, it called Deep Pyramid CNN**[10]**. In Resent, the Residual learning help the model easily get the better performance when CNN get deeper, and Deep Pyramid CNN borrows the idea from Resent but with pre-activation on Residual learning learning shortcut, and its number of filters keep unchanged. In this case, the CNN model with get more efficient learning and get more global information. Compared to the image classification problem, we do not need to care about the data length(size) problem which is easily to run out of GPU, because it is just 1D sequence.

I finally set the 1D CNN kernel size to be 4, in order to let the model get the enough n-gram information, but too large(bigger than 4) will let the model become more likely to overfit. Also, I add batch normalize layer and dropout layer to regularize the model, because the CNN model is six Residual blocks, and it is deeper than normal model.

6. Capsule

Actually this is in the last few days experiment. Thus, I did not spend most of the time at tuning this model. Also, I used the modified version, not the original Hinton's Capsule net [11]. The part of length layer is discarded here.

However, the result of it is really well, especially its private score and public score are really close. Seems the routing mechanism is the key part to let the model get the aggregated vectors (consensus) which is more stable than other models. Yet, base on my opinion, another key point is that Capsule GRU net has the similar structure as RCNN for text[15] after the GRU layers, that is, using CNN to extract the features, so it get deeper feature than normal RNN model.

• Other trick:

Before finding the good structure of model, the intuitive and efficient way is to use simple train test split method, shuffle the data and get the validation data for testing model's result, this can save computation cost. After that, I using 10 fold average to let model become more stable, without rely on one fold or simple split for validation.

Also, batch size has a little effect to the accuracy of the model's prediction. Actually, the paper**[9]** mentioned some of the evidence, that is, the bigger batch size with constant learning rate can get rid of gradient noise. In my experiment, seems having a few gradient noise will let some model be regularized and have better performance, but it is not very conclusive. On method which really works out is that if I get the good learning rate and want to get the bigger batch size to do more

efficient training, I will let the learning rate to be larger at the same time. This tip is in order to balance the weight decay made by the bigger batch size.

About the optimizer, I turn common the one, Adam, to Nadam, because in this competition Adam is easily to let model learning become really unstable and converge soon at the not most optimistic point. Thus, if we add Nesterov accelerated gradient to let the model learning be more sensitive to find the optimistic point? That is the Nadam**[12]**, actually it enhance my LSTM attention model most. This is because, Nesterov accelerated gradient can get a superior step direction by applying the momentum vector to the parameters before computing the gradient, not just depend on present momentum.

• Final ensemble:

For the diversity and saving computation cost, I only select five Neural network model for my consideration. At first, I use compute the Pearson correlation between each model's output, and if any outputs of two model's Pearson correlation are too larger (I set the threshold to 0.98), I'll use simple weight average to let it become a single model, that is multiple 0.5 for both of them.

At the final step, I use xgboost library for my second layer stacking. My tuning method base on that if I see the performance get more unstable or early stop happen too early, I'll set the model become less complex, so the main parameters I tune are max depth, the number of estimator, the subsample ratio for tree to train, and the subsample ratio of columns to constructing each tree. The key concept I did is to obey the "occam zrazor", because in this step is more likely to overfitting in the private leaderboard score.

• Final Conclusion:

Actually at the last's day public score our team are only at nearly top 6%, out of top 200, but due to robust stacking and the models' chosen strategy, we shake up in the top 5%. That is to say the 10 out of fold is stable really robust method to apply in the final ensemble.

Other competitor also transfer the data back to ASCII first. The feature engineering, including word embedding play the key role in this competition's final stage. Also, the new architecture really become the super star that can enhance the model performance. Like modified Capsule net or Deep attention RNN, it is very amazing that the performance of it dose not change a lot between public and private leaderboard.

In reality is that most of my model are not slow, but when training the model have long sequence like word level and char level multiple input model, the long sequence needs to spend nearly 5 hours.

Finally, I consider the efficiency of the whole model set and bring the simple concept: in order to keep the convenience of modification in the later day, so I use as less models to ensemble as possible, I think it is most suitable to the reality case, also my models are usually not very deep and their training speed are fast.

• Acknowledge:

Thanks for my friend lend the machine to extend my computation power, especially in this kind of competition depends a lot on Deep learning skills. Another thing is that in the last hour team up

with John Miller, mix up code with him, it improve my naive cross validation better. Also, his Light gradient boosting model give my stacking more diversity.

Appendix:
a.The performance on Kaggle leaderboard(public and private), Roc auc score:

- **Used model for ensemble:**

wiki.en.bin single gru:
public:0.9860, private:0.9847

wiki.en.bin and char word2vec single gru:
public:0.9859, private:0.9846

glove.840B.300D and char word2vec single gru:
public: 0.9855, private:0.9847

glove.twitter.27B.200D LSTM attention and skip connected channel:
public:0.9852, private:0.9845

crawl-300d-2M.vec DPCNN:
public:0.9847, private:0.9827

crawl-300d-2M.vec Capsule:
public:0.9847, private:0.9841

- **Ensemble:**

Final stacking:
public:0.9870, private:0.9867

b.Reference:
- **paper or report (Placed by the order of my schedule in this competition)**

1.Enriching Word Vectors with Subword Information
https://arxiv.org/pdf/1607.04606.pdf

2.GloVe: Global Vectors for Word Representation
https://nlp.stanford.edu/pubs/glove.pdf

3. Feed-Forward Networks With Attention Can Solve Some Long-Term Memory Problems:
http://colinraffel.com/publications/iclr2016feed.pdf

4.Hierarchical Attention Networks for Document Classification:
https://www.cs.cmu.edu/~diyiy/docs/naacl16.pdf

5.Convolutional Neural Networks for Sentence Classification:
https://arxiv.org/pdf/1408.5882.pdf

6.Fast And Accurate Deep Network Learning By Exponential Linear Units (elus)
https://arxiv.org/pdf/1511.07289v1.pdf

7. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm
https://arxiv.org/pdf/1708.00524.pdf

8. Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift
https://arxiv.org/pdf/1801.05134.pdf

9. DON'T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE
https://arxiv.org/pdf/1711.00489.pdf

10.Deep Pyramid Convolutional Neural Networks for Text Categorization
http://ai.tencent.com/ailab/media/publications/ACL3-Brady.pdf

11.Dynamic Routing Between Capsules
https://arxiv.org/pdf/1710.09829.pdf

12.Nadam report
http://cs229.stanford.edu/proj2015/054_report.pdf

13.Word2Vec using Character n-grams
https://web.stanford.edu/class/cs224n/reports/2761021.pdf

14. Website of Peter Norvig
https://norvig.com/spell-correct.html

15. Recurrent Convolutional Neural Networks for Text Classification
https://pdfs.semanticscholar.org/eba3/6ac75bf22edf9a1bfd33244d459c75b98305.pdf

d. Used pre-train word vectors and other open resource in the competition:

FastText:
1.wiki.en.bin
2.crawl-300d-2M.vec
Glove:
1.glove.840B.300D.txt
2.glove.twitter.27B.200D.txt

Attention:
1.issue on GitHub:
https://github.com/keras-team/keras/issues/4962
Capsule:
1.Kaggle kernel:
https://www.kaggle.com/chongjiujjin/capsule-net-with-gru

e. Device used in this competition and computation resource cost:

1.AWS p2.xlarge
2. PC with 4 core ,32 ram, and GTX 1080
* nearly spend more than 300 hour on doing experiments or practice
* 10 fold average of each model is nearly 2.5~4.5 hour for one model, base on the model's structure and batch size.