CS420 Course Project Report
Deep and Wide Learning in Text-Classification

Li Jiasen
ACM Honor Class, Shanghai Jiao Tong University
lijiasen0921@sjtu.edu.cn

Abstract

This paper introduces a text-classification model named deep and wide text classification. Applying this model, a weak sub-model can improve the overall performance. The wide linear model can effectively memorize the data using linear projections, while deep neural networks can generalize to previously unseen feature interactions.

1. Introduction

In real life, rating in a recommendation system is very important. Food market MeiTuan shows the satisfactory rate to the customers to assist them when choosing what to eat.

However, sometimes the service provider may be dishonest by cheating when rating. They might use programs to rate themselves very high.

We can collect the comments of customers and come up with a satisfactory rate, based on the natural language of the comments. And this can prevent the cheating I mentioned.

This paper will explain how to build a text-classification model to justify whether a comment is positive or not. The method this paper uses is deep and wide [Google 2016] in text-classification. By ensembling low bias models and low variance models, it can make improvement in performance even if the "wide" model is very weak.

2. Nature Language Pre-Processing

In this project, the data source we get is in the form of nature language Chinese.

In every line of the training data, there is a comment with a label telling whether the comment is positive or not. The unconcerned comments are treated as negative comments.

Chinese Sentences are made up of Chinese Words. Usually, 2 or 3 Chinese Characters form a Chinese Word. For example “喜欢” means “like”, and “不喜欢” means “hate”.

Here I used 3 ways to transform a Chinese Sentence into a Matrix. The methods are for different types of models.

2.1. Word Cut

It’s very natural to use words instead of letters in Language like English. Letter “a” in “apple” has nothing to do with “a” in “banana”. In Chinese, some words of different meanings also share the same Chinese Characters. So we are going to cut the words in Chinese and some words might be a hint whether a comment is positive.

However, Chinese Sentence has no space between words. When we see the sentence, we see Chinese Characters. So it’s not so easy to cut the words.

I used a word-cut library called jieba to cut the words.

Jieba is based on a dictionary in it. Jieba builds a Trie Tree to make quick scan of the sentence and see if there exists an ambiguity. Dealing with the ambiguity, Jieba can use its sentence dictionary to look up the word with the highest probability. [J Sun 2012]

After cutting the word, I turn the sentence into the form of “bags of words”. This type of data can be used by simple linear models which does not have too large Hypothesis Class H. Usually, such a simple model cannot cut the word by itself through the training, so it’s necessary to cut the words when pre-processing the natural language.

2.2. Directed Mapping

When training a complex model, word cut and bag of words may not be enough. If we just cut the words and put into the bag of words, the sequence information might be lost. So a better idea that might have less information lost might be to store the sequence of words in a sequence of one-hot vectors where every one-hot vector presents a word.

However, the word cut might not be always correct, and the number of the words is much larger than that of the characters. It’s better to store the characters instead of the words. A character is represented as a one-hot vector in dimension 2718, and 2718 is the size of the character dictionary in this problem.

There are at most 19 characters in a sentence, so 21 one-hot vectors form a matrix that represents a sentence. The first and the last one-hot vectors are zeros.

Character directed mapping can only be used by sequential models or models that at least can recognize character in different position of a sentence. That’s because if position information of a character cannot be taken, this may be equal to that the characters are mixed together, and reordered sentences cannot be recognized. For example, the sentence “我喜欢吃其它菜，但不喜欢这个” and “我不喜欢吃其它菜，但喜欢这个” are totally different, but if position is not considered, they are the same sentences.

2.3. N Gram

In most cases, the directly mapping method of pre-processing needs a complex model, such as CNN. However, if the given data is not enough to train such a complex model, a better choice might be to choose a "bag of words" that can preserve the sequential information.

N gram separates characters in two. And, every two neighboring is treated as a "word". This "word" is not a real word in nature language. But it can preserve much information. We can save the frequency of the words in a sparse vector. And we can have the sparse vector of frequency normalized so that the $\mathcal{L}_2$ norm of the vector is one. [Jonathan *et.al* 1982]

For example, sentence "不喜欢" can be saved as {(space, 不), (不, 喜), (喜, 欢), (欢, space) }. All the frequencies are $1/2$. The square sum is one. We can say "不喜欢" occurs very often if we find that "不喜" and "喜欢" occur very often. There might be counter-examples, but we ignore it.

After producing a large number of new "word"s, we only take those which occur in high frequency into account. Those which occur only once might not occur in the test set.

2.4. Data Augmentation

Data set can be enlarged by swapping several characters in a sentence. Some models are so large that cannot be easily trained with the given data. So I can modify each sentence a little bit, and form a new sentence. We assume that the new sentence can have the same label as the original sentence. This method is called shuffle in NLP. Other methods such as synonym replacement or translating back are a little bit difficult to implement without translation libraries, so we do not discuss here.

So what characters can be swapped when shuffling?

We can swap the neighboring characters because we assume that the way we humans read nature language does not care about the order of 2 or 3 characters we focus on.

For example, seeing "你喜吃欢什么", maybe you will not recognize this is reordered at first sight. Applying this method, a more powerful dataset can be formed.

3. Models

According to the corollary of the Theorem Occam's Razor in John E Hopcroft's book. [Blum, Hopcroft and Kannan 2015]

$$err_D(h) \leq err_S(h) + \sqrt{\frac{size(h)\ln(4) + \ln(2/\delta)}{2|S|}}$$

Here, S is the training set. D is the Ground Truth Distribution. The $size(h)$ denotes the number of bits needed to describe function h (classification model) in the given language. The $err_A(h)$ means the error rate of h, given the set A. This inequality shows that if we want $\delta_{err} \leq 10\%$, we should have $size(h) < 230$ approximately. And the more simple the model is, the less likely it would over-fit the model. Our mission is to apply one or more simple models and apply several complex models. The result of the models will be ensembled in the next Section.

I am going to introduce three models following, which I used in this project.

3.1. Linear Model

The Linear Model I used here is the Perceptron Model. The Sample Set S contains $|S| = n$ labeled examples $x_1, x_2, \ldots$ with shape $d \times 1$, where x_i is a vector saving the frequency of words.

We want to have vector $W_{d\times1}$ and vector $b_{1\times1}$, such that $W^T x_i > -b$ if and only if x_i is labeled 1 and $W^T x_i < -b$ if and only if x_i is labeled 0. We can say that $W_i x_i > 0$ is a evidence showing that the comment is positive. [Blum, Hopcroft and Kannan 2015]

We can say that the model is not likely to over-fit the data too much, because it's too simple.

How do we solve it? We solve it by gradient descent optimizer. You can see my repo for my handwritten gradient descent library in Python and the following algorithm in python.

Algorithm Gradient Descent for Perceptron Model

Input: $\boldsymbol{x_{d\times1}}, \boldsymbol{y_{1\times1}}$

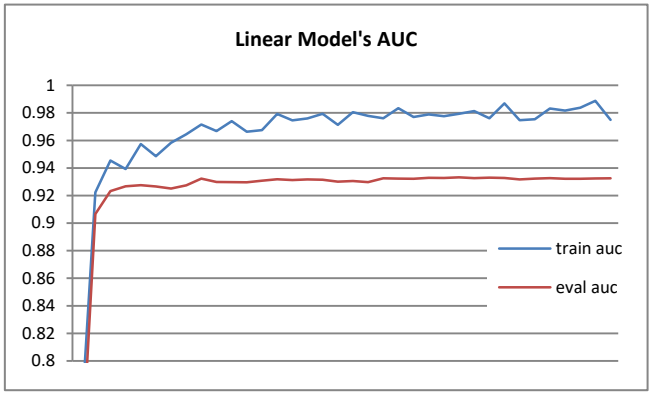
Output: $\boldsymbol{Predict}$

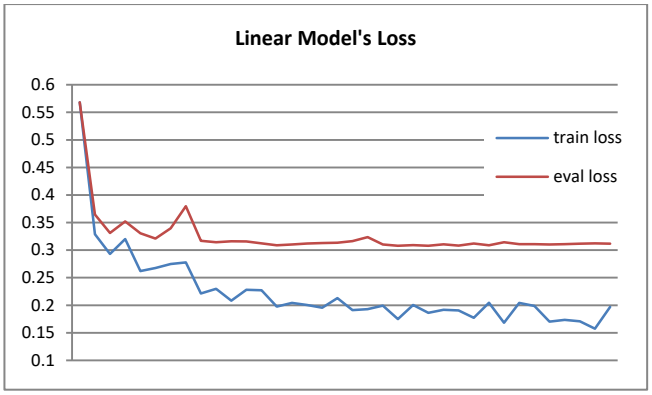
1. $W := Random_{d\times1}$
2. $b := Random_{1\times1}$
3. $for\ step\ in\ range(3000)$:
4. $\quad Predict := \frac{\exp(W^T x+b)}{\exp(W^T x+b)+1}$
5. $\quad W := W - l_{rate} \cdot \frac{\partial loss(y,Predict)}{\partial W}$
6. $\quad b := b - l_{rate} \cdot \frac{\partial loss(y,Predict)}{\partial b}$
7. $\quad (l_{rate}\ can\ decay\ here)$

We want to minimize the distance between the y and the prediction.

The loss function I used is cross entropy.

$$loss(y, pred) := -y\ln(pred) - (1-y)\ln(1-pred)$$





Here, we found that the result can be $evaluation\ auc \approx 93.3\%$, which is fine for Linear Regression.

3.2. CNN

In Linear Regression (Perceptron), the goal is:

$$minimize_{W,b}\ \Big(loss\big(y, f(W^T x + b)\big)\Big)\ \ (f(x) := \frac{e^x}{e^x+1})$$

The function $W^T x + b$ is so simple that it may not fit the Ground Truth very well in some cases.

We can use a more complex function instead to present the probability that a comment is positive. [ZHZhou 2016] We assume $x_{len\times d}$ is a data point, len is the max length of the sentence and d is the size of dictionary of characters.

Here we describe the function as an algorithm.

$(dim1 = 15\ \ dim2 = 30\ \ dim3 = 96)$

Algorithm CNN Function : Pred

Input: $x_{batch\times len\times d}$

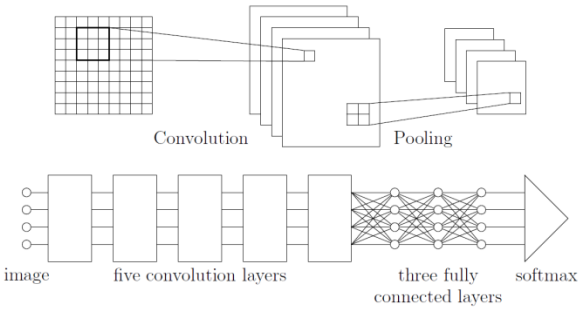
Output: $Predict_{batch\times d}$

1. $reshape\ x\ as\ x_{batch\times 1\times len\times d}$
2. $(using\ global\ variable\ W1_{1\times 3\times d\times dim1})$
3. $(using\ global\ variable\ B1_{dim1})$
4. $C1 := maxpool_{1\times 2}(relu(Conv2d(x, W1) + B1))$
5. $(using\ global\ variable\ W2_{1\times 3\times dim1\times dim2})$
6. $(using\ global\ variable\ B2_{dim2})$
7. $C2 := maxpool_{1\times 2}(relu(Conv2d(C2, W2) + B2))$
8. $reshape\ C2\ as\ C2_{batch\times len'}$
9. $(using\ global\ variable\ W3_{len'\times dim3})$
10. $(using\ global\ variable\ B3_{1\times dim3})$
11. $C3 := relu(C2\ W3 + B3)$
12. $(using\ global\ variable\ W4_{dim3\times 1})$
13. $(using\ global\ variable\ B4_{1\times 1})$
14. $C4 := C3\ W4 + B4$
15. $Predict := e^{C4}/(e^{C4} + 1)$

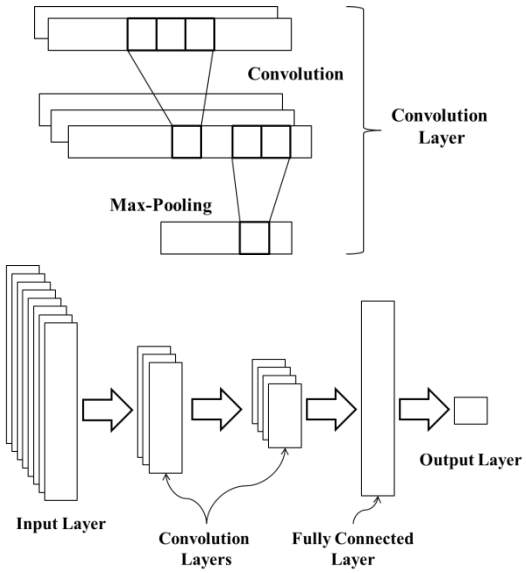
Matrix Broadcasting is assumed when describing the function above.

What is the function like?

We regard the sentence as an Image of shape $1 \times len$ with d channels in each pixel. So we can use CNN here as in Computer Vision.



The CNN in Computer Vision



CNN We use here

Line4 is the first layer. We want it to work as the word embedding, and we offer the neighboring information to each character and wishing it will help translating the character one-hot vector into a low-dimension vector. By max-pooling, words are roughly cut according to length.

Line 7 is the second layer. We want the words to share neighboring information in order to form phrases score and tell whether the phrases can indicate positive or negative emotion of the customer. Actually we cannot easily determine whether it is positive or negative by ourselves, but the next layer will recognize it.

The third layer in line 11 is a fully connected layer. Some phrase coming together may have special meanings. And using a fully connected layer can handle this to some extent. A subset of phrase can influence the result by occurring together.

The following layer is the output layer. We are not going to discuss here.

In CNN, I used the ADAM Optimizer in tensorflow to minimize the function. For the principle of the Optimizer, you can see my self-made python library Dreamer Flow in my repo, which is a little bit slower than tensorflow.

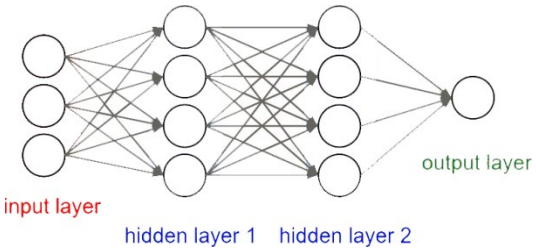
3.3. Multi-Layer Perceptron

Actually, if the data set is large enough, the CNN model can surely works well. However, we only have 16000 labeled data, including the evaluation data.

We are going to simplify the NN model.

We use the n gram with bag of words type of input to simplify the input layer, and this will not have too much information lost.

The input layer is the frequency of each word, and we have two fully connected layers and an output layer following the input layer.



In n gram, word embedding layer is not necessary and we don't care about the exact word.

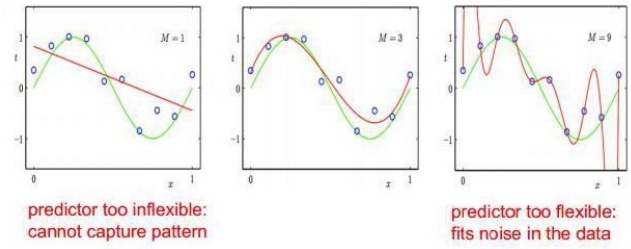
The first fully connected layer is designed to form the score of phrases according to the frequency of several words, because of the property of n gram. You can see the Section 2.3 for what's the property of n gram.

The second fully connected layer is designed to guess the emotion of customer according to a subset of phrases scores, and pass the answer to the output layer.

The function is much easier than CNN, and it is very similar to the CNN, so we are not going to describe it in the algorithm. See [Blum, Hopcroft and Kannan 2015] for details.

3.4. Dealing with over-fitting

Actually, you will not get the performance well enough if you didn't do anything with the over-fitting problem.



Over-fitting is a problem that your model can work on the training set very well but cannot work on the evaluation set well. This is often because of the bias of sampling of the training set. As shown above, our training data is just a sampling of the ground truth, which may have bias from the ground truth distribution. Over-fitting is mainly caused by the noise from ground truth to the training data. [Blum, Hopcroft and Kannan 2015] When we cannot add data set, what we can do is just to modify our model and decrease the flexibility. Sure we can simplify the model like what I did in Section 3.3, and other methods can also be applied to decrease flexibility.

Here, I am going to introduce several methods I used when I was dealing with over-fitting. Some other good methods such as early stopping is not used in the project.

3.4.1. Dropout

This works especially when I use the Neural Network in this project.

I randomly select some elements in each layer, and turn them into zeros. This can add variance and prevent local minimal to some extent.

In some way, we can regard dropout method as training a serious of models at the same time, and we can think that when we predict with this model, the answer is actually an ensemble answer. This idea is from the paper of Hinton. [Hinton *et.al* 2014]

In some way, we can regard dropout as adding noise to our training set, so that it is data augmentation to some extent. Some books say it is just like dividing points into "male" and "female" to add robustness of the model and avoid over-fitting caused by union of points that are not separable to some extent. [WNZhang 2019]

3.4.2. Regularization

Over-fitting can be seen as the model being too complex. We can use the "Occam's Razor" - Regularization - by penalizing complexity. It is because we think that the ground truth is elegant and not too complex. [ZHZhou 2016]

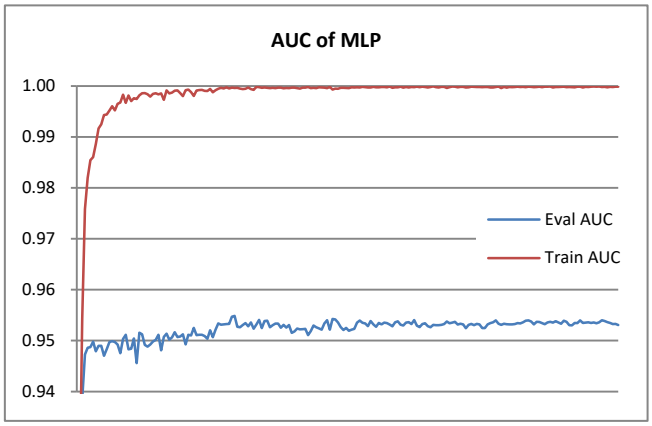
I used L2 regularization, and it just works better than L1 regularization. Maybe it's because a slight change is also allowed.

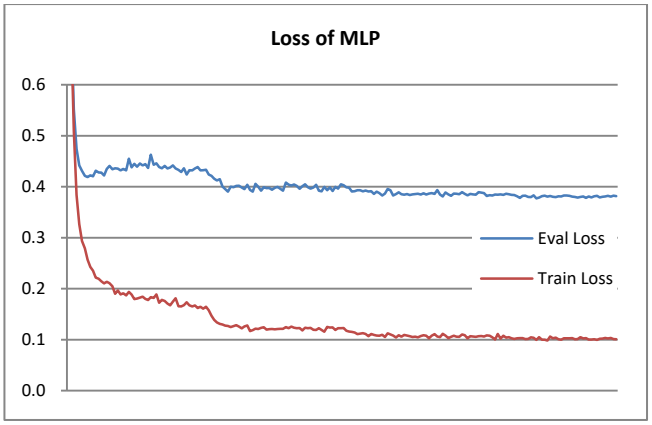
3.5. Learning Rate Decay

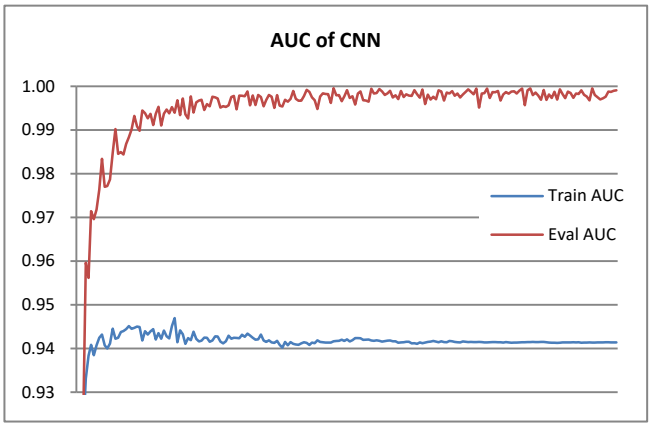
With Learning Rate Decay, it is less likely to fall into a local minimal of the function, because of the large Learning Rate at the beginning. It is going to have lower random err because of the lower Learning Rate at the end. [Zeiler 2012]

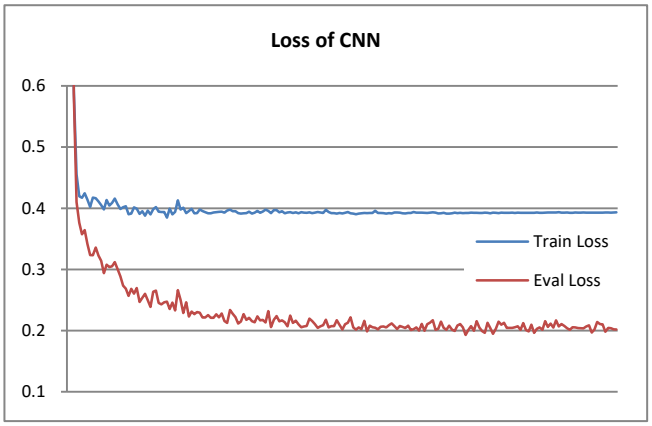
I cut the Learning Rate in half every 1000 steps. In order to choose any step number (e.g. n = 1000) to have Learning Rate Decay, I randomly shuffle the training set every step. So there is no episode or epoch here.

After fixing in 3.4 and 3.5, model in 3.2 and 3.3 works well.

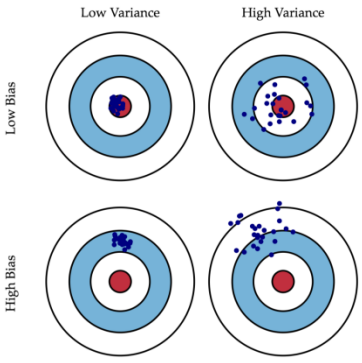








4. Deep and Wide Ensemble



Our training data is a sample from the ground truth distribution. The way we sample has variance from the ground truth. And when we train our model, we consider the training data is fixed, so we may say there is bias in the training data, and we will have bias in our model if it over-fits the training data. Truly ensemble is a way to deal with over-fitting, but it is not applied in a model, but between models.

Using ensemble, we can fix the "wide" linear model that is surely unlikely to over-fit the data with the "deep" NN models those are likely to, at least slightly, over-fit the data together. The ensemble model is going to have lower bias, because the linear model can pull the NN's over-fitting back. The ensemble model is going to have lower variance because it can have higher accuracy with the help of the NN model.

Using ensemble we average the models in some way we design. This way can be just weighted arithmetic mean, or any function else.

Here, I introduce my ensemble method.

4.1. Idea

I think that there are hints whether a comment is positive or negative. And hints can be valued by numbers.

$$f: x \mapsto y \quad Hint \to Probability$$

If the models produce hints, I can average the hints in some way to get the total hint, to get the total Probability of the comment being positive. We assume that there are three model to be ensembled.

$$Prob \approx f(average(h1, h2, h3))$$

$h1, h2, h3$indicate the hints provided by the models.

If the models produce the probability, I can just put an f inverse.

$$Prob \approx f\left(average\left(f^{-1}(p_1), f^{-1}(p_2), f^{-1}(p_3)\right)\right)$$

4.2. Choice of Function

So the choice of the function can be a question.

I choose the function $f(x) = \frac{e^x}{e^x+1}$, and I have $f(0) = \frac{1}{2}$, $\lim_{x\to+\infty} f(x) = 1$ and $\lim_{x\to-\infty} f(x) = 0$.

Using f function, the model would not be influenced by an uncertain $p = \frac{1}{2}$ too much. Actually, this is the softmax function with zero.

Assume that the first model is the "Wide" linear model.

I choose the function:

$$average(x, y, z) = \lambda_1 |x|^{1+\delta} \times sgn(x) + \lambda_2 y + \lambda_3 z$$

as the average function.

If $\delta > 0$, when the wide model is more confident, the wide model has a higher weight in order to have the possibility to pull the NN models back if they do not work well.

4.3. Choosing Parameters

I wrote a library to deal with this problem. First, I split the test into two parts. Then, I train the sub-models according to one part, and evaluate according to the other. Last but not least, I do Gradient Descent to decide the parameters according to the evaluation part of the data.

4.4. Experiments and Results

Model	Implementation	Pre-processing	Numbers	AUC
Linear	tensorflow	Word Cut (jieba)	6	0.93337
CNN	tensorflow	Directly mapped	4	0.94603~ 0.95121
Ensemble1	tensorflow	Linear & CNN	1	0.95735
MLP	tensorflow	N gram	4	0.95119
Ensemble2	tensorflow	Linear & CNN & MLP	1	0.96006

5. Conclusions

In this paper, we proposed deep and wide text classification, a deep and wide learning method for text classification, to overcome some limit of the classic models.

Wide linear models can effectively memorize the data using linear projections, while deep neural networks can generalize to previously unseen feature interactions.

It gains performance from these advantages: 0)deep and wide learning 1) adaptive weight for linear model in ensemble 2) special hint function ignoring those no-opinion models 3) using both MLP & CNN sub-model to increase stability 4) low requirement for large data thanks to the special use of n gram in MLP and the over-fitting dealing

Reference

[Blum, Hopcroft and Kannan 2015] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science.* Cornel 2015

[WNZhang 2019] WNZhang. *2019 CS420 Machine Learning Lecture.* SJTU 2019

[Bishop, C. M 1995] Bishop, C. M. *Neural Networks for Pattern Recognition.* Oxford University Press, NY 1995.

[ZHZhou 2016] ZHZhou. *Machine Learning.* Tsinghua University Press 2016.

[HLi 2012] HLi. *Statistical Learning Methods.* Tsinghua University Press 2012.

[J Sun 2012] J Sun. *Chinese word segmentation tool.* Github 2012.

[Google 2016] Google. *Wide & Deep Learning for Recommender Systems.* 2016.

[Jonathan *et.al* 1982] Jonathan J. Hull, Sargur N. Srihari *Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms* IEEE 1982

[Hinton *et.al* 2014] N Srivastava, G Hinton, A Krizhevsky. *Dropout: a simple way to prevent neural networks from overfitting.* Journal of Machine Learning Research. 2014.

[Zeiler 2012] Matthew D. Zeiler *ADADELTA: An Adaptive Learning Rate Method*