Large Scale Multi-label Text Classification With Deep Learning

Liang Xu
Fitme.ai
xul@fitme.ai

Abstract

Multi-label text classification is a more complex problem than single label text classification. Labels may have some internal relationship or hierarchical structure between each other. We investigate several neural networks for multi-label classification. Our baseline is fastText [1], a very simple model with n-gram features. We also explore two model with more complex structure: one is TextCNN [2], with multiple filters based on CNN; another is Hierarchical Attention Network [3], with hierarchical structure and attention mechanism.
These models are used for multi-label classification on Zhihu question & topic dataset. we show that deep models are more superior than shallow model.
F1 score for our baseline model is 0.72. The later two models is around 0.80, increase around 11% compare to shallow model. by using ensemble of latter two models [11], performance improves for another 1.0%.

1. Introduction

Multi-label text classification is a complex natural language processing task which require to predict multi-label for one or several sentences. Our baseline method is fastText. it is scalable fast and shallow method which can train large scale of data in a few minutes, even with lots of classes for the label. another feature of this method is that it use n-gram features as its input. so it can capture rich features. and the performance is fine.
However, this method is too simple, and may not complex enough or effective for modeling complex problem. to solve this problem, we explore convolutional neural network based model (TextCNN), and model with hierarchical structure and attention mechanism (Hierarchical Attention Network). from the experiment, we show that both models exceed the baseline more in a large margin.

2. Problem statement

One simple way to do multi-label task is to cast this problem into single label text classification. Suppose we have n labels for one sentence. then we can construct n pairs of sentence-label as training data; during inference, we can just retrieve top-k labels for a sentence. However, this setting has obvious problem: the training data is inconsistent or have logical contradiction. This is something like you tell the model this picture is cat in the first second, and one second later you tell that it is also a dog. It causes confuse to the model.
The second way to do multi-label text classification is to assume that each labels is independent.

given a sentence, the model is ask to maximize the prediction of each label independently. In another saying, for each label the possibility will be from 0.0 to 1.0; and the loss is based on cross entropy between true distribution of labels and prediction distribution of labels. In this essay we use the later method. we will now describe our models.

3. Models

3.1 Baseline model: fastText

It first embed each word in the sentence into distributed representation space, this word representations are then averaged into a text representation, which is in turn fed to a linear Classifier. it use softmax function to compute the probability distribution over the predefined classes. then cross entropy is used to compute loss.

This bag of word representation does not consider word order. in order to ease this problem, n-gram features is used to capture some partial information about the local word order; when the number of classes is large, computing the linear classifier is computational expensive. so it use hierarchical softmax to speed up training process. Blow is the diagram of fastText:

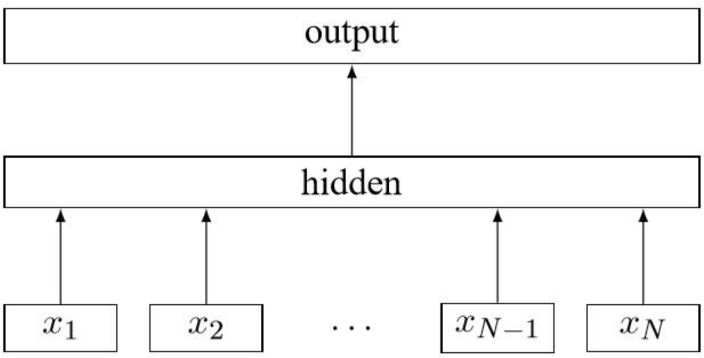
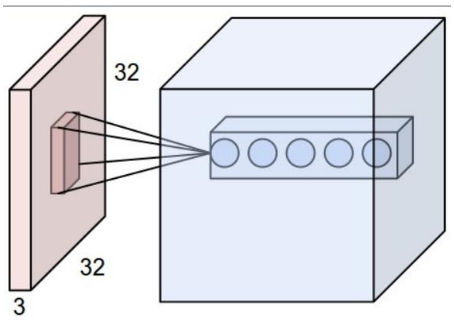


Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

3.2 TextCNN

Convolutional Neural Network is main building box for solve problems of computer vision. Now we will show how CNN can be used for NLP, in particular, text classification. Sentence length will be different from one to another. We use pad to get fixed length of sentence, that's say n. For each token in the sentence, we will use word embedding to get a fixed dimension vector, d. So our input is a 2-dimension matrix:(n,d). This is similar with image for CNN.

Firstly, we will do convolutional operation to our input. It is a element-wise multiply between filter and part of input(check below picture [15]). We use k number of filters, each filter size is a 2-dimension matrix (f,d). Now the output will be k number of lists, each list has a length of n-f+1, each element is a scalar. Notice that the second dimension will be always the dimension of word embedding. We are using different size of filters to get rich features from text inputs. And this is something similar with n-gram features.



Secondly, we will do max pooling for the output of convolutional operation. For k number of lists, we will get k number of scalars.
Thirdly, we will concatenate scalars to form final features. It is a fixed-size vector. And it is independent from the size of filters we use.
Finally, we will use linear layer to project these features to per-defined labels. You can see it from below:

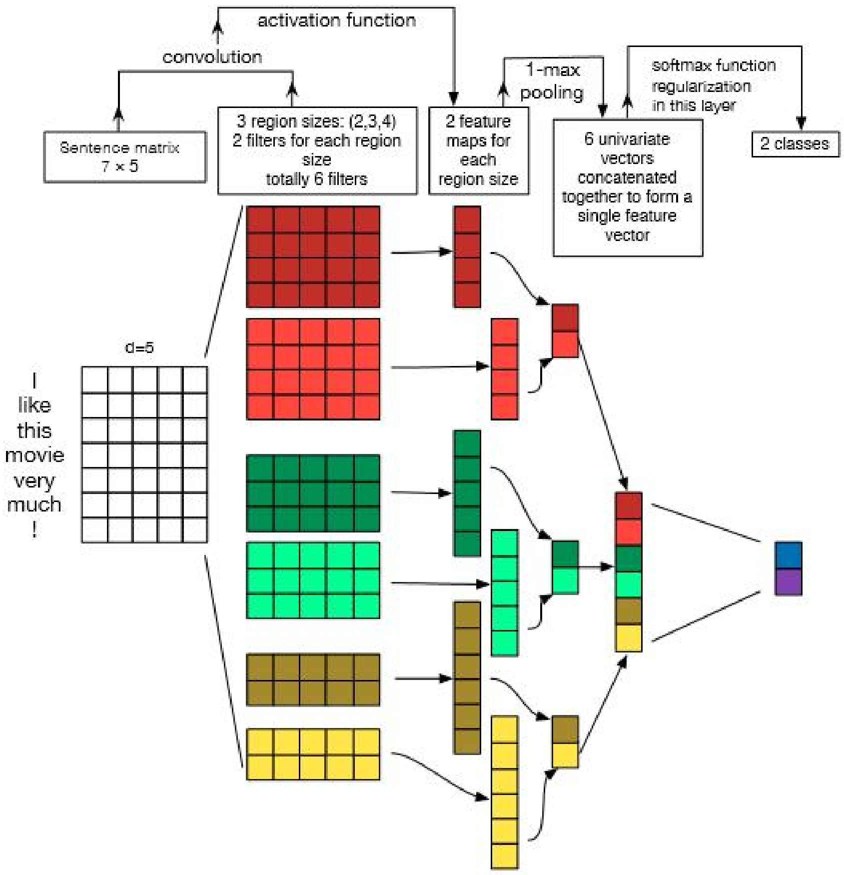


Figure2:
Illustration of a CNN architecture for sentence classification. Input sentence length is 7, dimension is 5. Input is 7*5. We use 3 filters, each region size is:2,3,4. two filters for each region size. Totally we have 6 filters. It do convolutional between input and filters, result in 6 lists. Length of each list is between 4 to 6. After max-pooling, we get 6 scalar numbers. We concatenate it to get a fixed-size(6) vector. we use a linear layer to get possibility distribution of labels.

3.3 Hierarchical Attention Network(HAN)

In NLP, text classification can be used for single sentence, but it can also be used for multiple sentences. we can call it document classification. Words are form to sentence, and sentence are form to document. In this circumstance, there may exists a intrinsic structure. How can we model this kinds of task? Does all parts of document are equally relevant? And how we determine which part are more important than another? Below is detail of this model.

It has two unique features:

1) it has a hierarchical structure that reflect the hierarchical structure of documents;
2) it has two levels of attention mechanisms used at the word and sentence-level. it enable the model to capture important information in different levels.

It has four part: word encoder; word attention; sentence encoder; sentence attention.

Word Encoder:

For each words in a sentence, it is embedded into word vector in distribution vector space. It use a bidirectional GRU to encode the sentence. By concatenate vector from two direction, it now form a representation of the sentence, which also capture contextual information.

$$\begin{aligned} x_{it} &= W_e w_{it}, t \in [1, T], \\ \overrightarrow{h}_{it} &= \overrightarrow{\mathrm{GRU}}(x_{it}), t \in [1, T], \\ \overleftarrow{h}_{it} &= \overleftarrow{\mathrm{GRU}}(x_{it}), t \in [T, 1]. \end{aligned}$$

Word Attention:

Some words are more important than another for the sentence. Attention mechanism is used. Instead of use additive attention [7,] or multi-head attenton [8], it use multiply attention. It first use one layer MLP to get u_{it} hidden representation of the sentence, then measure the importance of the word as the similarity of u_{it} with a word level context vector u_w and get a normalized importance through a softmax function. Word level vector is learned during training.

$$u_{it} = \tanh(W_w h_{it} + b_w) \quad (5)$$

$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)} \quad (6)$$

$$s_i = \sum_t \alpha_{it} h_{it}. \quad (7)$$

Sentence Encoder:

for sentence vectors, bidirectional GRU is used to encode it. Similarly to word encoder.

$$\begin{aligned} \overrightarrow{h}_{i} &= \overrightarrow{\mathrm{GRU}}(s_i), i \in [1, L], \\ \overleftarrow{h}_{i} &= \overleftarrow{\mathrm{GRU}}(s_i), t \in [L, 1]. \end{aligned}$$

Sentence Attention:

sentence level vector is used to measure importance among sentences. Similarly to word attention.

$$u_i = \tanh(W_s h_i + b_s), \quad (8)$$

$$\alpha_i = \frac{\exp(u_i^\top u_s)}{\sum_i \exp(u_i^\top u_s)}, \quad (9)$$

$$v = \sum_i \alpha_i h_i, \quad (10)$$

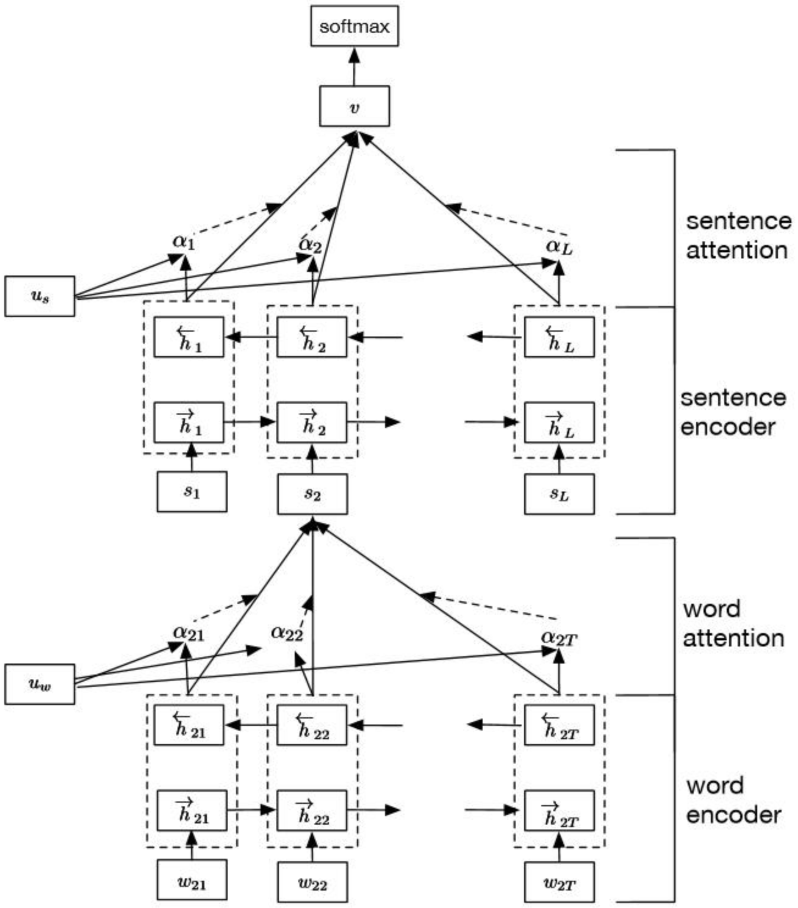


Figure3: Hierarchical Attention Network. Word level encoder and attention to capture contextual and important information in the word level; Sentence level encoder and attention to capture contextual and important information among sentences.

4. Experiments

4.1 Data Set

We use Zhihu question & topic challenge data set for our experiments [16]. According to wiki, Zhihu is a Chinese question-and-answer website where questions are created, answered, edited and organized by the community of its users. We were asked to training a model to automatically label the unlabeled data, given question and topic data set. It has 3 million questions, for each question it has one or more labels. The total labels is 1999. For each label it is corresponding to a

topic in Zhihu. There is a parent-children relationships between topics.
For the privacy concern, original questions and topics are not provided. Instead, all the name and description of questions and topics are replaced as symbol, something like ‘w111’,’c222’. It also provide word embedding at word and character level trained from large corpus.
217k questions without label is also provided as test set, for each question, we are asked to predict Top 5 topics.

4.2 Metrics
Performance is reported by positioned F1 score
Precision: the predicted label belong to one of true labels is considered to be right. However, the final accuracy is calculated by the weighted sum according to position.

$$Precision = \sum_{pos\ in\ \{1,2,3,4,5\}} \frac{Precision\ @\ pos}{log(pos+1)}$$

Recall: total right prediction of top 5 divide by total prediction of top 5.

Finally score is reported by:

$$\frac{Precision * Recall}{Precision + Recall}$$

As you can see that position of your prediction will also impact the performance. And the preceding label’s impact to the performance is large than later label’s impact.

4.3 Experiments
For all the experiments, we form input from four parts: word level of question name, character level of question name, word level of question description, character level of question description. We limit total length for the input, and limit length for each part.
we use pre-trained word embedding from word2vec in the training data set. Embedding size of word vector and hidden size of internal layer is set to 100. We split training data into training set(95%) and validation set(5%), performance is reported from test set with 217k data. Adam is used as optimizer. L2 regularization with lambda 0.01 to 0.001 is used. Learning rate is decay exponentially or decay by half whenever validation loss is not decrease.

Model	FastText(1)	TextCNN(2)	HAN(3)	Ensemble (2,3)
Performance	0.36	0.405	0.398	0.410

Figure 4:Performance of our models

4.3.1 baseline fastText
We’ve tested two version of fastText. One is from facebook. Another is implemented by our own. Performance is very close to each other, the former one is 0.362, the later one is 0.363. But the former one is faster. It only time around 10 minutes to train 3 million data, ours cost more than one hour. Instead of use hierarchical softmax loss, we use noise-contrastive estimation(NCE) loss.

We also test how n-gram features impact the performance. But in this data set, it has no remarkable change of performances between different n-gram features(uni-gram features, uni-gram to bi-gram, uni-gram to tri-gram).

4.3.2 TextCNN

This model has many filters with different filter size. And for a specific filter size, it use many filters to capture features with different aspects.

From the paper A Sensitivity Analysis of Convolutional Neural Networks[3] for Sentence Classification, we can see that many aspects can change the performance, but the main aspects to impact performance is the combination of filter size, and number of filters for each size. One key idea is to find single best filter size, then combine filter sizes near this single filter. For example, if single best filter size is 7, then optimal choice may be filter size(6,7,8). The number of filters for each size usually below 512.

After run many experiments, we got our best combination of filter size is:[3,4,5,7,10,15,20,25], filter size is 512. Usually single best filter is not small, could be something like 7 or less. Because of total sentence length for our input is 100, which is a somehow a little long, so single best filter is longer in our case. Use big filter number, it is very computational expensive. If you want to do quick experiment, you can just try small filter number, like 128.

Compare to other model, TextCNN is not prone to over-fit, since it is based one single layer, and with features from horizontal direction, it is highly paralleled.

4.3.3 Hierarchical Attention Network(HAN)

We construct hierarchical structure of input data to suit for this model. As usually, we will feed a sentence to the model, but internally we will split the sentence into 4 parts to form multiple sentences. Now we can use word level and sentence level encoder and attention to model the input.

One special thing for this model is that it can learn very fast. In other words, it can converge very quickly. Sometimes, by training only a very few epoch, it can reach very low validation loss and good accuracy. We suspect that attention mechanism help to detect the prominent features, and hierarchical structure enhance this strength.

However, this model is very prone to over-fit. Sometimes when loss not decrease and suddenly it become big and big. We had to use big batch size, and carefully choose init value for the parameters.

4.3.4 Ensemble

We also try ensemble our models to form a better model. In our experiment, we use a simple style of ensemble: weight sum of logits of different models is used to get final logits. The weight is based on single model’s performance. then it is used to make a prediction. By ensemble our two model, the performance improve around 1.0%.

5. Discussion and Conclusion

We explored several models to do test classification, and explained the main parts of these models. We also talked our finds though the experiments. Compare to very simple model like fastText, more complex model like TextCNN and HAN can capture more complex structure of task, and thus performance is much better.

One flaw of these models is that all these models did not use label information to make a prediction, and relationship of labels is undiscovered.

Currently character based CNN, and very deep CNN get state of art performance in many NLP tasks, like text classification. Due to lack of character information of this data set, we did not able to try these models yet.

All of codes is available in our repository [14]. It has many more models available for text classification, you can check it if you like.

Reference

1.Bag of Tricks for Efficient Text Classification

2.Convolutional Neural Networks for Sentence Classification

3.A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

4.Deep Learning for Chatbots, Part 2-Implementing a Retrieval-Based Model in Tensorflow, from www.wildml.com

5.Recurrent Convolutional Neural Network for Text Classification

6.Hierarchical Attention Networks for Document Classification

7.Neural Machine Translation by Jointly Learning to Align and Translate

8.Attention Is All You Need

9.Ask Me Anything:Dynamic Memory Networks for Natural Language ProceSUSUssing

10.Tracking the state of world with recurrent entity networks

11.Ensemble Selection from Libraries of Models

12.Large Scale Multi-label Text Classification with Semantic Word Vectors

13.CS224D FinalProject: Neural Network Ensembles for Sentiment Classification

14.https://github.com/brightmart/text_classification

15.http://cs231n.stanford.edu/

16.https://biendata.com/competition/zhihu/