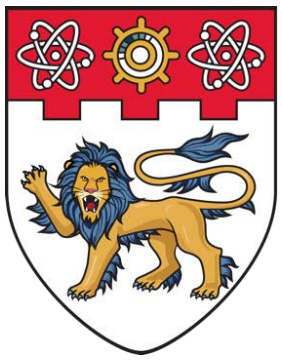
Deep learning for sentence/text classification

Yu, Rongqian

2018

http://hdl.handle.net/10356/76043



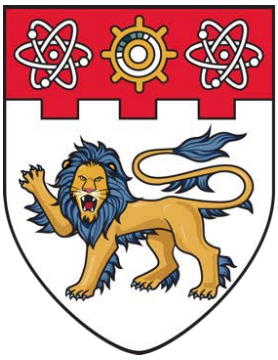
NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Deep Learning for sentence/text classification

YU RONGQIAN

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

2018



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Deep Learning for sentence/text classification

YU RONGQIAN

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER CONTROL AND AUTOMATION

2018

Table of Contents

Nanyang Technological University Ingenuity Statement

I hereby declare that the dissertation (design) thesis submitted by me is my own research work conducted under the guidance of my supervisor and the research results I have obtained. To the best of my knowledge, except where specifically noted and acknowledged in the text, the dissertation does not include research published or written by others, nor does it include any degree or certificate used to obtain a Nanyang Technological University or other educational institution material. Any contribution made by my comrades working with me to this research has been clearly stated in the paper and gratefully acknowledged.

Signature:_____

Date: _____

Acknowledgements

In the course of this experiment, I first thank Professor Suganthan for his guidance and provided me with a workplace and hardware support. I am very grateful to Dr. Rakesh for answering some professional questions for me.

At the same time, I feel that all the people on the network who have given me help have selflessly answered some professional questions for me.

Thanks to my parents for supporting me at any time and providing financial support for completing my graduate degree.

YU RONGQIAN

2018/3/28

Acronyms

CNN	Convolution neural network
NLP	Natural Language Processing
DL	Deep learning
AI	Artificial intelligence
RNN	Recurrent neural network

List of Figures

List of Tables

Abstract

Deep Learning for sentence/text classification

G1701110E YU RONGQIAN

Supervisor P.N. Suganthan

Deep Learning Architectures have been achieving state-of-the-art results in many application scenarios. Particularly, the performance of Deep Convolution Neural Networks (Deep ConvNets) in computer vision tasks is incontestable. The wave of ConvNets is sweeping through other applications other than vision tasks. There are some instances of ConvNets used for Natural Language Processing (NLP) tasks such as sentence/text classification. The objective of this project is applying Deep Learning models such as Recurrent Neural Networks, ConvNets for sentence/text classification tasks and suggest ways to improve their performance.

In this design, I used CNN(Convolution neural network) network structure as my framework, using python3 programming language and PyTorch deep learning tool to complete the preparation of the software and experiments on the remote server in the laboratory to get the final result(using GPU acceleration).

Key words: Sentence/text classification, Deep Learning, Convolution neural network（CNN), Natural Language Processing (NLP), python, PyTorch , remote server, GPU acceleration, VDCNN.

Chapter 1
Introduction

1.1 Background

With the rapid development and popularization of Internet technology, how to classify, organize and manage vast data sets has become an important research topic. In these data, the text data is the largest category. Text classification has a wide range of applications, for example:

The news site contains a large number of articles, based on the content of the article, the articles need to be automatically classified according to the subject (for example, automatically divided into political, economic, military, sports, entertainment and so on.

In the e-commerce website, the user conducted the transaction behavior after the classification of goods, businesses need to be divided into positive and negative evaluation of the user to obtain feedback on the statistics of the various users of the product.

E-mail frequently receive spam messages, text classification technology from a large number of spam messages to identify and filter, improve the efficiency of mailbox users.

The media has a large number of daily submissions, relying on text classification technology to automatically review the article, marking submissions of pornography, violence, politics, spam and other irregularities.

Prior to the 1990s, the dominant text categorization method has been a knowledge-based approach: with the help of professionals, a large number of inference rules are defined for each category, and if a document satisfies these inference rules, one can determine Belong to this category. However, this method has obvious shortcomings: the quality of the classification depends on the quality of the rules; the need for a large number of professionals to formulate the rules; there is no extendibility, different areas

need to build a completely different classification system, resulting in the development of resources and A huge waste of resources.

The machine learning technology can solve the above problems well. Based on the statistical theory, the algorithm is used to make the machine have a similar human-like automatic "learning" ability - to get the law through the statistical analysis of the known training data and then use the law Predict and analyze unknown data. The basic process of using machine learning methods in text categorization is as follows:

Labeling - use a manual classification of a batch of documents as a training set (for machine learning materials);

Training - computer dig out some of these documents The rules that can be effectively classified generate the classifier (the set of rules summarized);

Classification - The generated classifier is applied to the document set to be classified to obtain the classification result of the document. Because machine learning has a good practical performance in the field of text categorization, it has become the mainstream of this field.

1.2 History and problems to be solved

Prior to Hinton's proposed Deep Belief Network (DBN) in 2006, neural networks were extremely complex and difficult to train functional networks, so they could only be studied as a mathematical theory.

Before neural networks became a powerful machine learning tool, classical data mining algorithms had many successful applications in natural language processing. We can use some very simple and easy to understand models to solve common problems, such as spam filtering, part of speech tagging.

But not all problems can be solved with these classic models. Simple models do not accurately capture the subtleties of language, such as satire, idioms or context.

Abstract-based algorithms, such as the pouch model, are not effective when extracting the sequence properties of text data, and n-grams are severely challenged by the "curse of dimensionality" problem when simulating a generalized situation. The

impact of HMM model is limited by Markov nature and it is difficult to overcome the above problems.

These methods also have applications in more complex NLP problems, but they have not achieved very good results.

The first technological breakthrough: Word2Vec

Neural networks can provide semantically rich word representations and bring a fundamental breakthrough to the NLP field.

Prior to this, the most commonly used characterization method was one-hot coding, where each word was converted to a unique binary vector with only one non-zero entry. This method is seriously affected by sparseness and cannot be used to represent any word with a specific meaning. Word Representation Projected into Two-dimensional Space in Word2Vec Method.

However, we can try to focus on a few surrounding words, remove the intermediate words, and predict the surrounding words by entering an intermediate word on the neural network. This is the skip-gram model; or on the basis of the surrounding words, the prediction of intermediate words, ie Continuous Word Bag Model (CBOW). Of course, this model is useless, but it turns out that it can be used to generate a powerful and efficient vector representation, while preserving the semantic structure of the word.

Further improvement: Although the Word2Vec model outperforms many classic algorithms, there is still a need for a solution that captures the long- and short-term order dependency of text. For this problem, the first solution is the classic Recurrent Neural Networks, which uses the temporal nature of the data and uses the previous word information stored in the implicit state to transfer each word to the train in an orderly manner. In the network.

There are many tasks involved in the interaction between the computer and the human language, which may be a simple matter for humans, but it brings a lot of trouble to the computer. This is mainly caused by subtle differences in language, such as satire, idioms and so on.

In terms of complexity, the following lists several NLP areas that are still in the exploratory phase:

The most common area is Sentiment Analysis, which may be the simplest. It usually comes down to determining the speaker/author's attitude or emotional reaction to a particular topic. This emotion may be positive, neutral and negative.

Generates multiple active neurons in the conversation network. It is clear that even with unsupervised training, the network can distinguish different emotional categories.

We can apply this method to Document Classification, which is an ordinary classification problem, rather than playing several labels for each article.

Next, we will introduce a truly challenging area - Machine Translation. This is a completely different area of research from the previous two tasks. We need a predictive model to output a sequence of words instead of a tag. In the study of sequence data, the addition of deep learning theory has brought a huge breakthrough to this field.

We may also want to build an automatic text summarization model, which needs to extract the most important parts of the text while preserving all the meanings. This requires an algorithm to understand the full text, and at the same time it can lock specific content in the article that can represent most of the meaning. In the end-to-end approach, the Attention Mechanisms module can be introduced to solve this problem.

The last area is Question Answering, which is an extremely relevant research direction for artificial intelligence. The relevant models not only need to understand the issues raised, but also need to fully understand the concerns in the text and accurately know where to look for answers.

Since deep learning provides the appropriate vector representation for various data (such as text and images), you can use different data characteristics to build different models.

So, there is a Visual Question Answering study. This method is relatively simple, you only need to answer the questions based on the given image. The task is simple enough to sound like a seven-year-old, but the deep model cannot output any reasonable results without supervision.

Chapter 2
Process of Sentence/text classification

2.1 Traditional text classification methods

Text categorization is a classic problem in the field of natural language processing. The earliest related researches can be traced back to the 50s of the last century when it was classified by the pattern of experts. Even in the early 1980s, it was developed to use knowledge engineering Expert system, the benefits of doing so is to quickly solve the top problem, but obviously the ceiling is very low, not only time-consuming and laborious, the coverage and accuracy are very limited.

Later, along with the development of statistical learning methods, especially the increasing number of Internet texts online and the rise of machine learning disciplines since the 1990s, a set of classical games for solving large-scale text categorization problems has been gradually formed. The main routine of this stage is the Artificial Character Engineering + Shallow classification model. Training text classifier process see below:

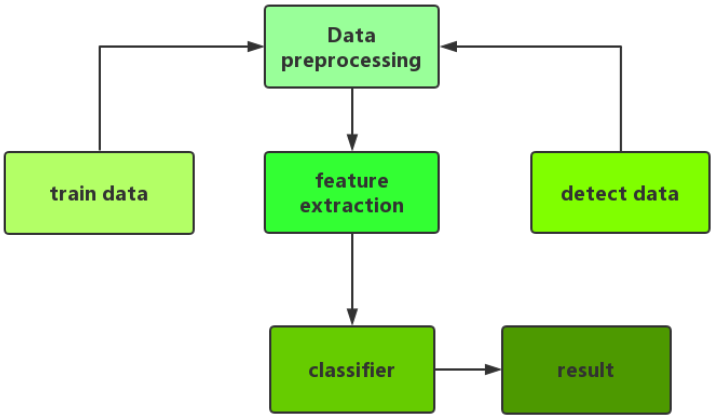


Figure 2.1 Object recognition process

2.1.1 Feature Engineering

Feature engineering is often the most time-consuming and labor-intensive in machine learning, but extremely important. In abstract, machine learning is the process of transforming data into information and then refining it into knowledge. The characteristic is the process of "data-> information", which determines the upper limit of the result. The classifier is "information-> knowledge" Process, it is to approaching this limit. However, the feature engineering is different from the classifier model, does not have strong versatility, often need to combine the understanding of the characteristics of the task.

The natural language domain where the text classification problem lies naturally also has its unique characteristic processing logic. Most of the tasks of the traditional sub-classification are also here. Text feature engineering sub-text preprocessing, feature extraction, text representation of three parts, the ultimate goal is to convert the text into a computer-readable format, and encapsulate sufficient information for classification, that is, a strong feature of expression.

2.1.2 Data preprocessing

The process of text preprocessing is the process of extracting keywords from the texts to represent the texts. The Chinese text processing mainly includes two stages: the text segmentation and the stop words. The reason for the segmentation is that many researches show that the characteristic granularity is that the granularity of the word is much better than the granularity of the word, which is actually well understood because most classification algorithms do not consider the word order information, and obviously too much "n-gram" information is lost based on the word granularity .

In all computer vision algorithms, preprocessing is almost always the first step, with the goal of maximizing the appearance of each image (such as color information, Light and dark information, size information, etc.) as much as possible. Data preprocessing refers to the processing of raw data. By performing certain operations on the data, the data can be better prepared for subsequent operation analysis. In this process, in fact, the input data is optimized, when an algorithm or a class of database experiments, often need to preprocess the data.

The preprocessing of the raw data is often related to the sampling device to be dealt with. Taking the license plate recognition as an example, if the license plate is to be identified, the alphanumeric information on the license plate should be extracted and then segmented and identified. After the completion of these operations, the numbers and letters can be identified, showing the necessity and importance of data preprocessing. However, some data bring their own characteristics already have the ideal nature, it is not necessary for data preprocessing, this situation is not very common.

In the area of data preprocessing, there are two things that need to be done:

(1) high-quality data mining needs the same high-quality data to support;

(2) If the data is not pre-processed, then the original data not only can not be effective, but also the training process will have a significant impact. For example, the original data will often have incomplete data, relatively high noise values, etc., if the input to the system without the untouched data, there will be a lot of interference.

In the field of data preprocessing, the methods that can be used can be divided into the following categories:

(1) noise reduction operation: delete the noise in the original data;

(2) Extracting specific content: Extracting certain specific content for certain specific requirements;

(3) Sampling: Sampling represents the subset in the raw data;

(4) Standardization: for specific input requirements, often in order to meet the follow-up algorithm.

2.1.3 Text representation and feature extraction

The purpose of the textual representation is to convert the preprocessed text into a computer-understandable way, which is the most important part of determining the quality of the text. The traditional approach is to use the BOW (Bag Of Words) or Vector Space Model. The biggest drawback is that the textual context is ignored. Each word is independent of another and can not represent the semantic information. An example of the bag model is as follows:

(0, 0, 0, 0, ... , 1, ... 0, 0, 0, 0)

In general, the amount of thesaurus is at least a million, so the word bag model has two biggest problems: high latitude and high sparsity. The bag-of-words model is the basis of the vector space model. Therefore, the vector space model reduces the dimension through feature selection and increases the density through the feature weight calculation.

Vector space model text representation of the feature extraction corresponding feature selection and feature weight calculation of two parts. The basic idea of feature selection is to rank the original feature items (terms) independently according to a certain evaluation index, select some feature items with the highest scores, and filter out the remaining feature items. Commonly used evaluation documents frequency, mutual information, information gain, χ^2 statistics.

The main feature of eigenvalue is the classic TF-IDF method and its extension method. The main idea is that the importance of a word is proportional to the word frequency in the category and inversely proportional to the number of appearances of all categories.

2.1.4 Semantic-based text representation

The traditional approach to text representation in addition to the vector space model, there are semantic-based text representation, such as LDA theme model, LSI / PLSI probabilistic potential semantic index method, the text that these methods are generally believed that the text can be said that the deep representation of the document, The word embedding text distributed representation is an important foundation for deep learning methods, the following will be demonstrated.

2.1.5 Classifier

Most of the classifiers are statistical classification methods. Basically, most machine learning methods are applied in the field of text classification, such as Naïve Bayes, KNN, SVM, maximum entropy and neural networks, Traditional classification model is not the focus of this article, here is not expanded.

2.2 Depth learning text classification method

The above text introduces the traditional text classification approach. The textual representation of the main problems of the traditional approach is that high latitude and high sparseness, the ability of expressing features is very weak, and the neural network is not good at processing such data. In addition, the feature engineering, High cost. However, one of the most important reasons why deep learning initially achieved great success in images and speech is that the original data of images and speech are continuous and dense, with a local correlation. Depth learning to solve large-scale text classification problem is the most important solution to text representation, and then use CNN / RNN and other network structures to automatically obtain the ability to express features, remove the complex artificial features engineering, end to end to solve the problem. The following will be introduced:

2.2.1 Distributed representation of text: word embedding

Distributed Representation In fact, Hinton was first proposed in 1986, the basic idea is to express each word into n-dimensional dense, continuous real vector, as opposed to one-hot encoding vector space is only one dimension is 1 , The rest are 0. The biggest advantage of distributed representation is that it has a very powerful feature representation capability. For example, each dimension of n-dimensional vector k values can represent k ^ {n} concepts. In fact, both the hidden layer of neural network and the probabilistic theme model of multiple potential variables apply distributed representation. Below is the 2003 Bengio network structure in A Neural Probabilistic Language Model:

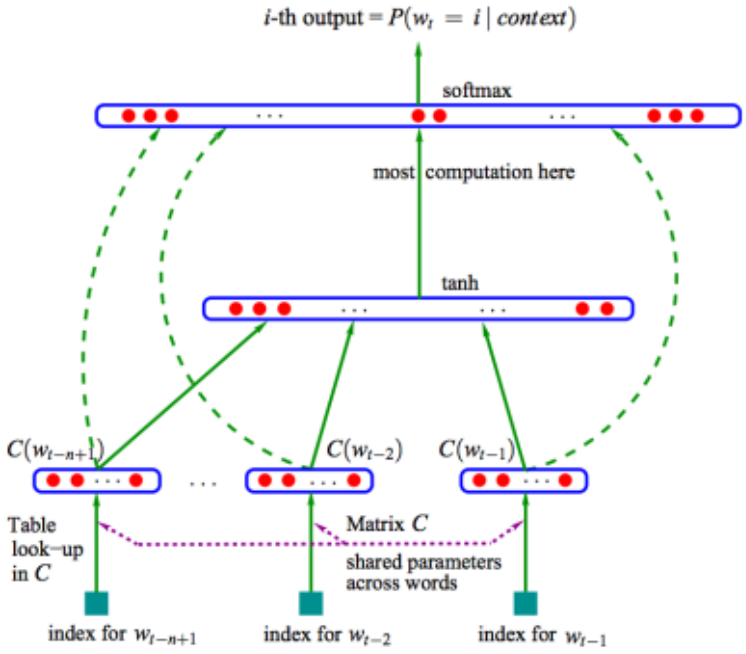


Figure 2.2 A Neural Probabilistic Language Model[1]

Although Hinton proposed the distributed representation of words in 86 years, Bengio proposed NNLM in 2003. The real vector of word vectors is the two Word2vec articles published by google Mikolov in 13 years. Efficient Estimation of Word Representations in Vector Space and Distributed Representations of More importantly, it released the easy-to-use word2vec toolkit, which has been well verified in the semantic dimension and greatly promoted the process of text analysis. The figure below is the structure of CBOW and Skip-Gram proposed in this paper, which is basically similar to NNLM. The difference is that the model removes the non-linear hidden layer and has different forecasting targets. CBOW is the predicate of the current word context word, Skip-Gram is the opposite.

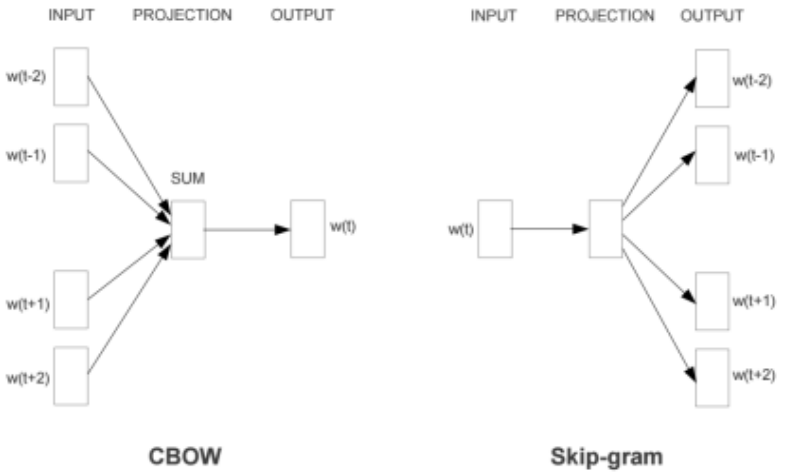


Figure 2.3 CBOW and Skip-Gram[2]

At this point, the representation of the text turns the text data into a continuous dense data similar to the image and the voice through the difficult way of high-sparseness and high-sparse neural network through the expression of the word vector. Deep learning algorithm itself has a strong data migration, many of the previous deep learning algorithms that are applicable in the field of image such as CNN can also be well moved to the field of text, the next section specifically describes the depth of learning methods in the field of text categorization .

2.2.2 Deep Learning Text Classification Model

The word vector solves the problem of text representation, and the text classification model introduced in this section solves the problem of automatic feature extraction (ie feature representation) by using CNN / RNN and other deep learning networks and their variants.

fastText is mentioned above word2vec author Mikolov fought Facebook after 16 years in July just published a paper Bag of Tricks for Efficient Text Classification. Putting the fastText here is not because it is the mainstream approach to text categorization, but rather it's simple and straightforward:

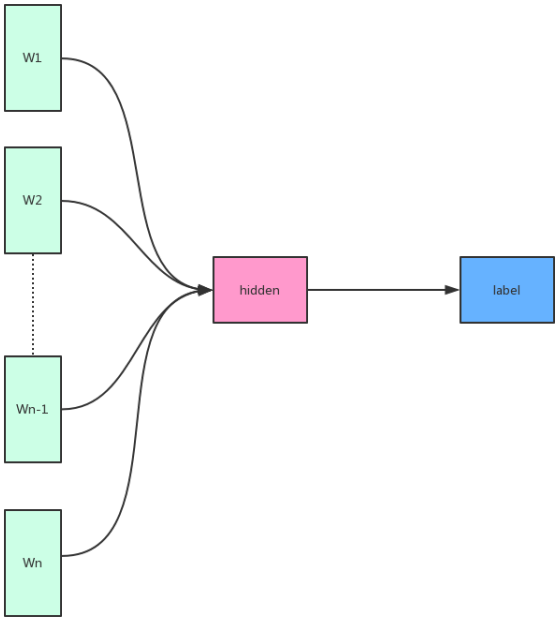


Figure 2.4 fastTest[3]

The principle is to average all the word vectors in a sentence (in a sense it can be understood that there is only one avg pooling special CNN), and then directly to the softmax layer. In fact, the article also added some n-gram features trick to capture local sequence information.

TextRNN: Although TextCNN can perform well in many tasks, one of the biggest problems with CNN is the fixed filter_size field of view. On the one hand, it is impossible to model longer sequence information. On the other hand, the oversized parameter adjustment of filter_size is very tedious. The essence of CNN is to do the textual representation of the features, while the recurrent neural network (RNN), which is more commonly used in natural language processing, can better convey the contextual information. Specifically, in the text classification task, Bi-directional RNN (actually using bi-directional LSTM) can be understood in a sense as capturing "bi-directional" n-gram information with variable length.

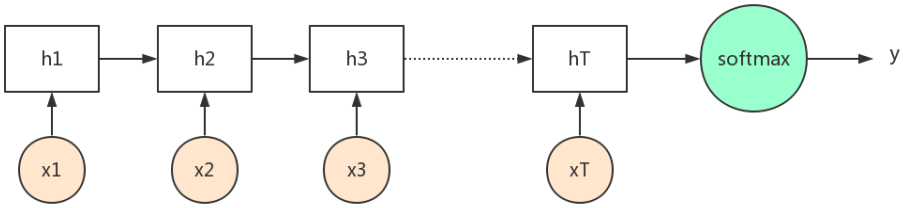


Figure 2.5 LSTM schematic diagram

RNN is a very standard network in the field of natural language processing. It has applications in many scenarios such as sequence tagging / naming convention / seq2seq model. In this paper, RNN is introduced for classification problem Design, the figure is LSTM schematic diagram for the network structure, the example is the result of the last word directly connected layer softmax output.

The method used in this article will be elaborated later

Chapter 3
Convolution neural network

3.1 history of development

The history of the Neural Network dates back to 1962. In an earlier study of cat visual cortical cells, Hubel and Wiesel proposed the notion of receptive field. In 1984, according to the above concept, a Japanese scholar named Kunihiko Fukushima put forward the concept of neocognitron, which is the first time that the concept of receptive field has been applied to artificial neural networks (ANN, Artificial Neural Network Network) can also be seen as the first network implementation of convolutional neural networks, the main application of the direction of the handwritten numeral recognition. This was followed by Yan LeCun, the father of the convolutional neural network mentioned earlier, to construct the first convolution neural network architecture in the strict sense. And his proposed network architecture is also called the LeNet-5 network architecture. Now, LeNet-5 is still the most typical convolutional neural network architecture, which consists of three major components: convolutional layer, pooling Layer, all connected layer. LeNet-5 In simple terms, the working principle is this: The convolution layer can cooperate with the pooling layer to form a number of volume units, and the convolution group is used to extract the features layer by layer. This step can be said to be affected by Feel the inspiration of the wild. Then through the full connection layer, to complete the final classification.[4],[5]

Since then, deep learning studies have been quiet until 2012, when AlexNet made its name on the ImageNet Image Sorting Contest of the year. In general, there are about 1000 possible categories for imagenet images. In each image, the computer can make five prediction results at the same time. No matter which one is accurate, the prediction is accurate. On the other hand, if all five results Is wrong, even if the failure, this classification error rate is called top5 error rate (top-5 error rate). And AlexNet reduced the top-5 error rate by a full 10 percentage points, and then convolved the neural network back to people's perspectives.

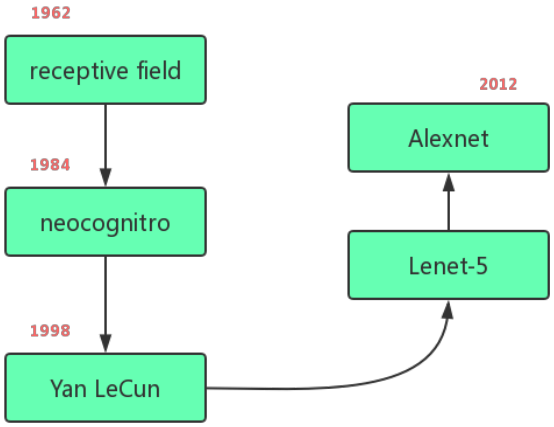


Figure 3.1 convolutional neural network development history

3.2 overall network model structure

With the progress and development of deep learning technology, with the help of convolutional neural networks, the field of image recognition has made great breakthroughs in recent years. The major machine learning laboratories in the world are also slowly beginning to pay attention.

The convolutional neural network algorithm has highly invariant properties to many kinds of deformations, such as tilt and translation of images. This algorithm belongs to the part of artificial neural network and plays an extremely important role in speech recognition and image processing. Convolutional neural network algorithm with weight sharing network structure, this special network structure similar to the biological neural network. Network input is graphics, images, voice, video and other information, and in the traditional neural network, the need for complex feature extraction work, and to carry out data reconstruction process. The advantages of convolutional neural networks become more pronounced when multi-dimensional images are input, significantly reducing time costs.

The traditional network identification model uses an artificially designed feature extractor to collect valid information of input data while shielding other information to obtain a set of feature vectors that can be input to a Classifiers trained to achieve

classification. Convolutional neural network network model is very different, this network structure combines feature extraction and classification, you can directly enter the original image, the feature extraction process hidden in the network, the direct output of the classification results.

Convolutional neural network is a further improvement on backpropagation neural network (BPNN, Figure 3.2), the common features are:

(1) Forward calculation of output value;

(2) reverse the weight and bias adjustment.

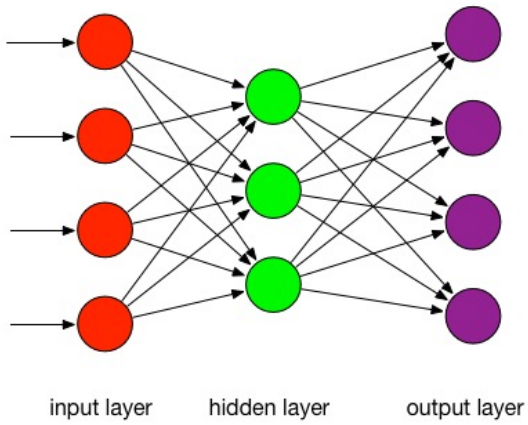


Figure 3.2 Backpropagation neural network

The biggest difference is that the perceived domain of a neuron in a convolutional neural network is a partial neural cell in the upper layer but not all the neural cells. That is, the neural cells in the adjacent layers are partially connected, and the standard backpropagation neural network is adjacent Nerve cells in the layer are fully connected.

Convolutional neural network architecture has the following three recognized important aspects:

(1) Local Area Awareness: In a standard backpropagation neural network, all the pixels are chaotic, the connection between the pixels is not utilized, and the local perception can find a line on the picture or Is a dot, these are the local characteristics of the data, and then by the combination of these local basic features, you can form the overall vision.

(2) Weight Sharing Mechanism: As shown in the following figure, there are several maps in the network diagram. Each layer of the network is composed of several figures, each figure contains several neural units. These neural units All share the same convolution kernel, where the convolution kernel can also be understood as a weight, which tends to represent a feature. For example, as mentioned above, if a line on a picture is taken as a Convolution kernel, convolution kernels are convoluted with the whole picture by using the knowledge of convolution calculation. The probability that a convolutional kernel has a large probability of one line is high while that of other lines is less likely. This is particularly similar to convolution operations when we use the fixed-size weight to match the image. For a standard backpropagation neural network, the convolution kernel is not a specific feature, but rather all the weights of this layer, also called overall perception. The weight sharing mechanism can effectively reduce training parameters.

(3) Spatial or temporal sampling: When a feature is found, we are concerned with the relative position of other features rather than specific locations. For example, identify the Chinese word "好". When we identify the "女" After the word, you just need to know that when the word "子" is to the right of "女", you can be sure that it is "好". As for where this word is in the whole image, it is not necessary for identification, so For distorted images or distorted images, this disregarding the specific location of the identification strategy is significantly better than the traditional identification methods. So to sample, to ignore the specific location of the feature.

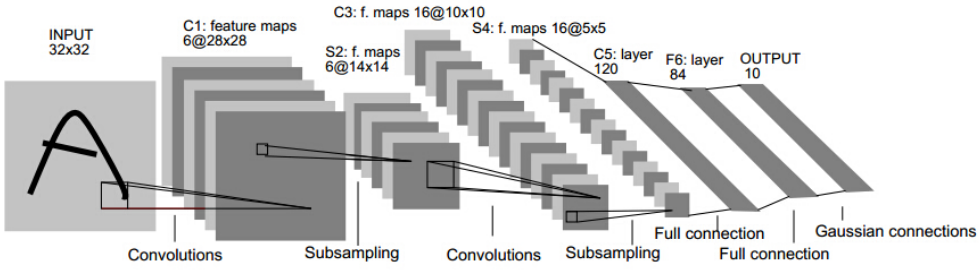


Figure 3.3 LeNet-5 Convolutional Neural Network Diagram[6]

When there is spatial distortions in the input data, such as distortion of the image data, or changes in the input data in time series, the above three characteristics of the

convolutional neural network can make the recognition maintain strong robustness and stability Sex. The general idea of convolution neural network is to alternately set the sampling layer and the convolution layer. Through the convolution layer, the features can be extracted and then the features of the pictures can be further described through a certain combination. Figure 3.3 is an example of a LeNet-5 convolution neural network.

3.3 Traditional LeNet-5 CNN

Here I use the LeNet-5 network to elaborate further on the calculation process and the input-output mapping relationship in the convolutional neural network.

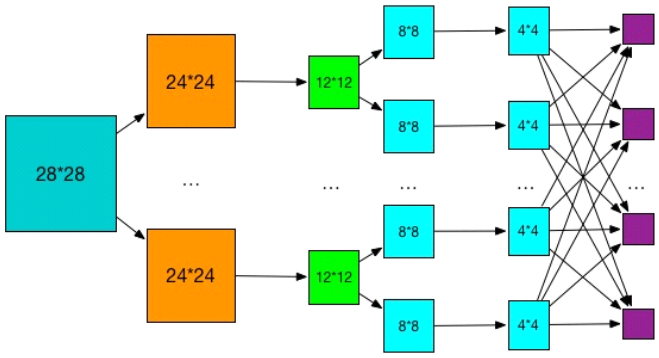


Figure 3.4 LeNet-5 CNN architecture

3.3.1 Forward transmission calculation

Input layer:

The input layer is not a value, but an input vector. In order to reduce the complexity of the system, a grayscale image is generally used. When the input is an RGB color image, it needs to be normalized in advance if it is normalized to [0, 1], the use of sigmoid activation function, of course, can also be used tanh tangent trigonometric function, the difference between the two is that when the input between [1,1], sigmoid function value changes are very sensitive, when close to the boundary or When it exceeds the boundary of the interval, it will enter the saturation state, that is, lose the required sensitive performance, which will greatly affect the accuracy of the network prediction; while the input of the tanh tangent trigonometric function has a very

good nonlinear monotonous rise and nonlinear monotone The descent is very suitable for the gradient descent algorithm used in this system, which will be mentioned later. At the same time, the fault tolerance rate is higher. However, the saturation period is relatively delayed compared to the sigmoid activation function.The size of the picture is the size of the vector. From Figure 3.4, the input is a 28*28 matrix.

Convolutional layer:

As can be seen from Fig. 3.4, the input of the convolution layer C1 comes from the input layer, and the input of the convolution layer C3 comes from the sampling layer S1. Therefore, in general, the input of the convolution layer comes from two parts, one is the input layer. One is the sampling layer. As mentioned in the weight sharing section, in the convolutional layer, each map has an equal-sized convolution kernel. In LeNet, the convolution kernel size of the input layer to the convolution layer is 5*5. For a more intuitive presentation, an example is shown in Figure 2.5. For the sake of simplicity, only 4*4 maps and 2*2 convolution kernels are used. After the convolution operation, we can get a 3*3 characteristic matrix. The calculation process is shown in Equation 3-1.

$$(4-2+1)\times(4-2+1)=3\times3 \qquad \text{Equation 3-1}$$

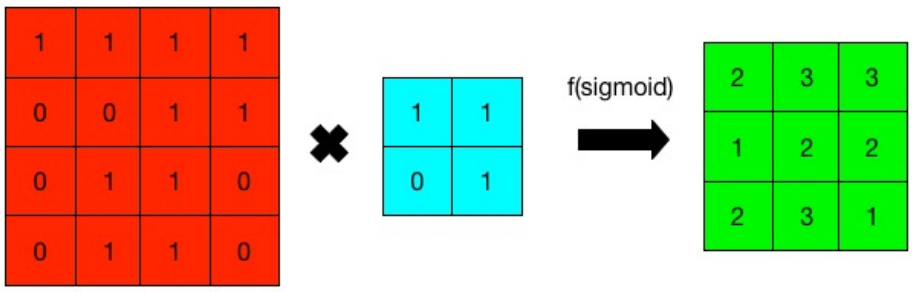


Figure 3.5 Convolutional process

This is equivalent to the convolution kernel traversing once on the output map of the previous layer, so the final feature map is to accumulate the corresponding elements after the convolution operation, add a bias, and finally perform the sigmoid operation.

The formula is shown in Equation 2-2. Assume that the output of the upper layer is Mj, the convolution kernel is Kj, and the output is M2j.

$$M2j = sigmoid(sum(Mij \times kij) + bias) \qquad \text{Equation 3-2}$$

A special point to note is that the number of plots of the convolutional layer is determined during the initialization process. However, in the convolutional layer, the size of the plot depends on two aspects. The size of the plot of the convolution kernel and the previous input layer is specified. The relationship is as shown in Equation 3-3. Assume that the convolution kernel is a k-order square matrix. The map input by the previous layer is an n-order square array false. Then the map size of the convolutional layer is the c-order square matrix.

$$c \times c = (n-k+1) \times (n-k+1) \qquad \text{Equation 3-3}$$

Subsampling layer, Pooling layer:

The specific task of the down sampling layer is to sample the previous layer, that is, to use Aggregation statistics in the adjacent small areas. After such processing, useful information can be effectively extracted and the amount of calculation can be reduced. As shown in Figure 3-6, the size of the region is scale*scale. In this example, the average of the small regions used, the size of the convolution kernel is 2*2, and the values of all elements in the convolution kernel are 1/4. In the implementation, the situation will change, sometimes using the maximum value of a certain area. It should be noted that the convolutional calculation window has overlapping, as shown in Figure 3.6, the deep color labeling area, but the calculation window used is not overlapped, so we need to remove the overlapped portion and get the output of the sampling layer:

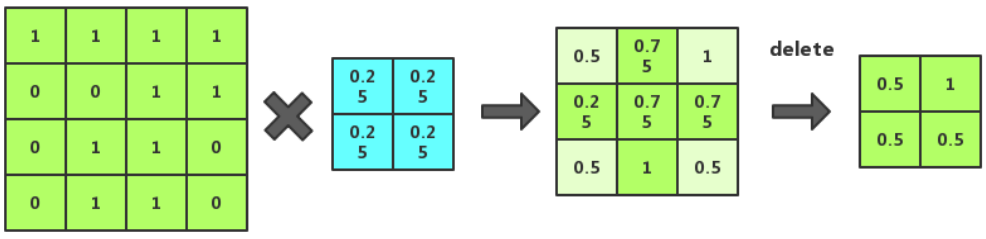


Figure 3.6 Sample layer diagram

3.3.2 Reverse Transfer Adjustment Weights

Similar to the back propagation process of standard backpropagation neural networks, convolutional neural networks also adjust the weights and offsets by minimizing the residuals, which is also one of the most complex places in convolutional neural networks. The structure determines the structure processing methods are also different. At the same time, due to the existence of weight sharing, the task of calculating the residuals is not simple.

Output Layer Residual:

Similar to the standard backpropagation neural network, in the convolutional neural network, the residual of the output layer is calculated differently from the middle layer, and the residual of the output layer is calculated as Equation 3-4. The error value is the output value and the class label. The value gets:

$$\delta_i^{(n_i)} = \frac{\partial}{\partial z_i^{(n_{(i)})}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_i)}) \cdot f^1(z_i^{n_i})$$

Equation 3-4

Convolutional residuals (when the next layer is the sampling layer):

First, assuming that the residue of the sampling layer has been calculated, it is now necessary to add a layer of convolution layer residuals. As can be seen from Figure 3-4, the size of the map of the sampling layer S2 (S4) is the convolution layer C1 (C3). 1/4 of the calculation process is shown in Equation 3-5.

$$\frac{1}{4} = \frac{1}{scale \times scale} \quad when\ scale = 2$$

Equation 3-5

At the same time, it can be seen that the number of maps in the two layers is the same. In order to keep the residual of the sample layer consistent with the map output by the previous layer, the sample layer residual and the size matrix of the scale*scale Perform Kronecker product calculations and expand with this operation. Figure 3.7 shows the expansion process and the calculation process.

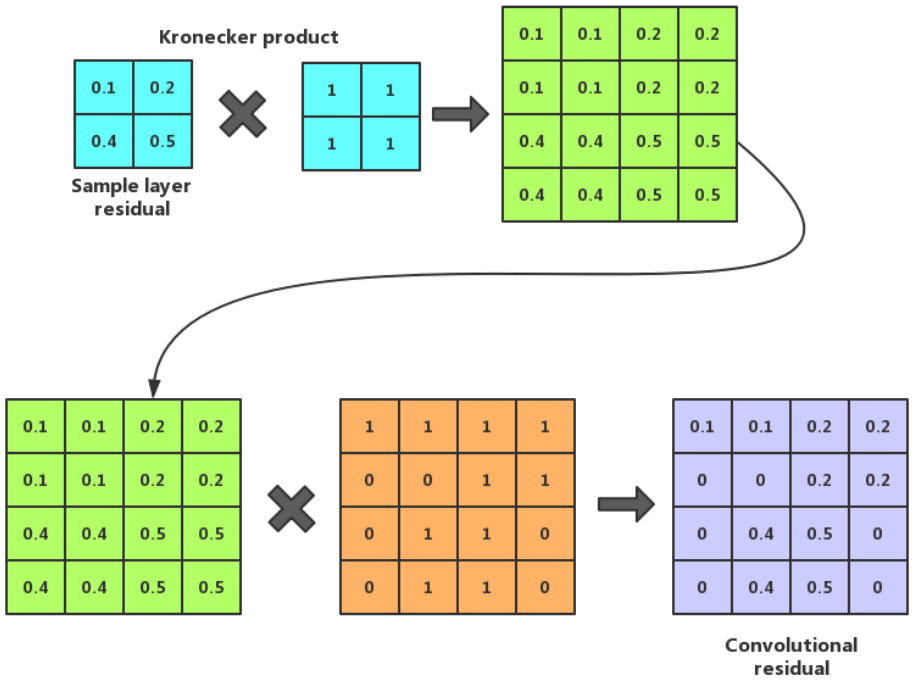


Figure 3.7 Expansion and Computation of Convolutional Residuals

3.3.3 Convolutional Neural Network Training Process

Before training, the two concepts are first defined. There are guided learning and non-directed learning. In the field of pattern recognition, the current mainstream direction is directed learning, but in cluster analysis, the more commonly used It is an unguided learning method.

In the guided learning method, the sample category is known, and the natural distribution tendency is not the standard for dividing space. Instead, the classification boundary is determined according to the separation degree of the sample and the heterogeneous sample of the same type of sample, and different types of The sample is divided. This learning process is long and complex and requires constant adjustment of the boundary location.

Convolutional neural networks can do this. Without any input/output mathematical expressions, they can learn massive mappings. They only need to use

existing models to train convolutional neural networks. Mapping capabilities are inherent in the network.

Therefore, the convolutional neural network uses a guided learning method, and its sample set is composed of vector pairs similar to the input vector and the ideal output vector. These samples can all be collected from a system that is running on the Internet. Before training, all weights of the system need to be given a random value. This random value is called "small random number". If the weights are all equal, the network cannot train. Different random numbers can guarantee the convolutional neural network. The normal training process. At the same time, the small random number can also prevent the network from entering the saturation state because the weights are too large, which makes the training fail.

The convolutional neural network training algorithm includes four steps. The four steps can be divided into the following two phases:

Forward stage:

1. Select a single sample (X, Yp) randomly and enter it into the network.
2. Calculate the corresponding actual output, denoted as Op.

At this stage, information moves through the layers, from the input layer to the output layer. After the network finishes training, this process is performed when it is running normally. In this process, it is calculated by Equation (3-6):

$$Op = F_n(\dots(F_2(F_1(XpW^{(1)})W^{(2)})\dots)W^{(n)}) \quad \text{Equation (3-6)}$$

Backward propagation stage:

1. Calculate the difference between the actual output Op and the ideal output Yp;
2. Using the above-mentioned minimization error method, adjust the weights.

The work of the backward propagation stage requires strict control accuracy. Here, the error of the p-th data in the network is defined as Ep. Then, the error of the entire data set is shown in Equation 3-7.

$$E_{all} = \sum Ep \quad \text{Equation 3-7}$$

See Equation 3-8 for the total error:

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(y_{pj} - o_{pj})^2 \qquad \text{Equation 3-8}$$

Because this phase propagates in the opposite direction of normal propagation, it is called the backward propagation phase. Sometimes this phase is also referred to as the error propagation phase, because in the forward propagation, only the error of the output layer can be calculated, while other errors require the output layer error to be inversely calculated. The next section will introduce the process of convolutional neural network training. Define the variables as per Table 3-1:

Table 3-1 Convolution Neural Network Training Variable Definition

letter	The meaning of the representative
N	The number of units entered
L	The number of units in the middle layer
M	The number of units in the output layer
X	Network input
H	Intermediate output
Y	The actual output of the network
D	Target output
V_{ij}	Weight from output unit to hidden layer unit
W_{jk}	Hidden layer to output layer weight
θk	Output layer unit threshold
φj	Hidden layer unit threshold

We can get that the middle layer unit outputs the following Equation 3-9,

$$h_j = f(\sum_{I=1}^{N-1} V_{Ij} x_I + \emptyset_j) \qquad \text{Equation 3-9}$$

The output layer unit output is as follows Equation 3-10,

$$y_k = f(\sum_{j=1}^{L-1} W_u h_j + \emptyset_k) \qquad \text{Equation 3-10}$$

Activation function following Equation 3-11,

$$f(x) = \frac{1}{1+e^{-kx}} \qquad \text{Equation 3-11}$$

The training of the network can be as follows:

1. Determine the training sample set and randomly select 500 data from it as the training set. Initialize V_ij, W_jk, θk, and φj as small random numbers (nearly 0), while giving an initial value of the precision control parameter ε and the learning rate α;
2. Enter X from the training set to the network and set the D value;
3. Using Formula 3-9, H is obtained, and Formula 3-10 is reused to obtain Y;
4. Comparing y_k and d_k, the output error (M) is calculated by Equation 3-12;

$$\delta^k = (d_k - y_k)\ y_k(1-y_k) \qquad \text{Equation 3-12}$$

The intermediate layer error (L) is calculated by Equation 3-13;

$$\delta_j = h_j(1-h_j)\sum_{k=0}^{M-1}\delta_k W_{jk} \qquad \text{Equation 3-13}$$

5. Calculate weights and threshold changes according to Equation 3-14 to Equation 3-17;

$$\Delta W_{jk}(n) = \frac{\alpha}{1+L} * (\Delta W_{jk}(n-1)+1) * \delta_k * h_j \qquad \text{Equation 3-14}$$
$$\Delta V_{ij}(n) = \frac{\alpha}{1+N} * (\Delta V_{ij}(n-1)+1) * \delta_k * h_j \qquad \text{Equation 3-15}$$
$$\Delta\theta_k(n) = \frac{\alpha}{1+L} * (\Delta\theta_k(n-1)+1) * \delta_k \qquad \text{Equation 3-16}$$
$$\Delta\emptyset_j(n) = \frac{\alpha}{1+L} * (\Delta\emptyset_j(n-1)+1) * \delta_j \qquad \text{Equation 3-17}$$

6. Adjust the weights and thresholds according to Equation 3-18 to Equation 3-21;

$$W_{jk}(n+1) = W_{jk}(n) + \Delta W_{jk}(n) \qquad \text{Equation 3-18}$$
$$V_{ij}(n+1) = V_{ij}(n) + \Delta V_{ij}(n) \qquad \text{Equation 3-19}$$
$$\theta_k(n+1) = \theta_k(n) + \Delta\theta_k(n) \qquad \text{Equation 3-20}$$
$$\emptyset_j(n+1) = \emptyset_j(n) + \Delta\emptyset_j(n) \qquad \text{Equation 3-21}$$

7. Whenever the value of k changes to M, the total error needs to be determined according to Equation 3-22. If the error requirement is satisfied, continue to the next step; if it is not satisfied, return to the third step;

$$E = \frac{1}{2}\sum_{k=0}^{M-1}(d_k - y_k)^2 \qquad \text{Equation 3-22}$$

8. End the training and save the weights and thresholds so that the subsequent training can be called at any time.

This time we use an improved convolutional neural network structure.

3.4 TextCNN

The structure of TextCNN proposed in this article in 2014 is selected for this article (see below). The network result of fastText does not consider the word order information at all, and it uses the n-gram trick trick to explain the importance of local sequence information.

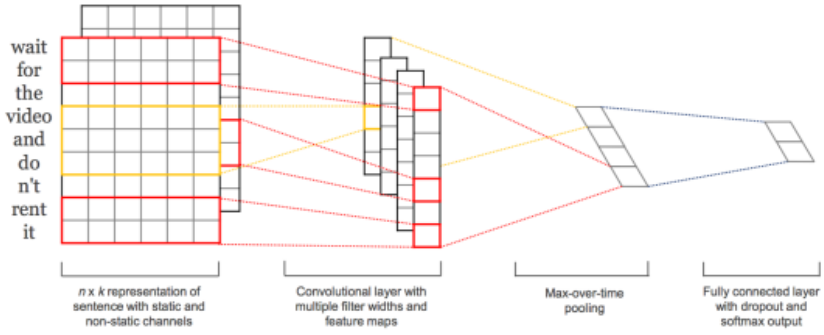


Figure 3.8 TestCNN architecture[7]

TextCNN detailed process: The first layer is the leftmost 7 by 5 sentence matrix, each line is a vector of words, the dimension = 5, this can be analogy for the original pixels in the image. Then through a one-dimensional convolutional layer with filter_size = (2,3,4), there are two output channels for each filter_size. The third layer is a 1-max pooling layer, so that different lengths of the sentence after the pooling layer can become fixed-length representation, and finally connected to a fully connected softmax layer, the probability of each type of output.

Features: The feature here is word vectors, both static and non-static. The static method uses word vectors such as word2vec pre-training, the training process does not update the word vectors, and essentially belongs to the migration learning. Especially when the amount of data is relatively small, static word vectors tend to work well. Non-static is to update the word vector in the training process. The recommended method is the fine-tunning method in non-static, which initializes the word vectors based on the

pre-train word2vec vector, and adjusts the vector during training to accelerate the convergence. Of course, if there is sufficient training data and Resources, direct random word vector initialization is also possible.

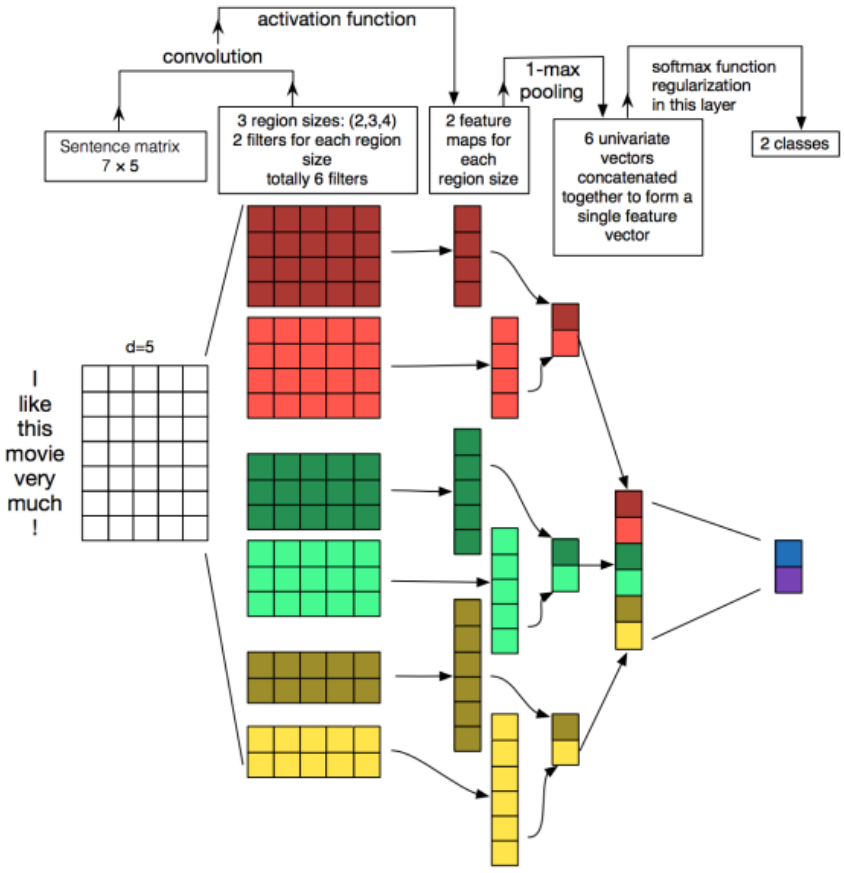


Figure 3.9 TextCNN detailed process diagram [8]

Channels: (R, G, B) can be used as different channels in an image, and channels for text input are usually embedding methods (such as word2vec or Glove). In practice, static word vectors and fine- Tunning the word vector as a way to do different channels.

One-dimensional convolution (conv-1d): the image is two-dimensional data, the text expressed by the word vector is one-dimensional data, so in the TextCNN convolution is a one-dimensional convolution. The problem with one-dimensional convolution is the need to design filters of different widths for different filter_size filters.

Pooling layer: The use of CNN to solve the problem of text classification is still a lot of articles, such as this A Convolutional Neural Network for Modeling Sentences

The most interesting input in the pooling into (dynamic) k-max pooling, pooling stage to retain the k largest information , Retains the global sequence information. For example, in the emotional analysis of the scene, for example:

"I think the scenery here is not bad, but there are too many people"

Although the first half shows that emotions are positive, the global text expresses negative emotions, and k-max pooling can capture such information well.[9]

Chapter 4
TextCNN Code analysis

4.1 The overall structure

The TextCNN class builds the most basic CNN model with an input layer, a convolutional layer, a max-pooling layer, and a final output softmax layer.
But because the entire model is for text (rather than the traditional processing object of CNN: images), some minor adjustments have been made correspondingly in the operation of CNN:

1. For text tasks, the input layer naturally uses the word embedding for the input data representation.
2. Next is the convolutional layer. The convolution kernels that we often see in image processing are square, such as 4*4, and then convolve along the width and height of the entire image for convolution operation. But the “image” input in nlp is a word matrix, such as n words. Each word is represented by a 200-dimensional vector. This “image” is a matrix of n*200. The convolution kernel has only been slid in height. The width is the same as the dimension of the word vector (=200). That is to say, every time the window is swept, it is a complete word. It does not convolve a part of the word "vector". This also guarantees the word. The rationality of the smallest granularity in the language. (Of course, if the granularity of the study is character-level rather than word-level, it needs to be handled in another way)
3. Since the width of the convolution kernel and the word embedding are the same, a convolution kernel for a sentence, the result of the convolution is a vector, shape = (sentence_len - filter_window + 1, 1), then, after the max-pooling It's a Scalar. Therefore, this is also a difference from the image convolution, and it needs attention.
4. It is precisely because of the max-pooling just to get a scalar, in the nlp, will implement multiple filter_window_size (such as the width of 3,4,5 words,

respectively, as the convolution window size), each window_size has num_filters (such as 64) Convolution kernels. A convolution kernel gets only a scalar and is too lonely. Smart people combine the num_filter scalars convolved with the same window_size to make up the feature_vector under this window_size.

5. Finally, all the feature_vectors in the window_size are also combined into a single vector, as the input to the last layer softmax.

4.2 parameters

Since the TextCNN class is based on YoonKim's thinking, then one of our next important step is to sort out the various parameter settings mentioned in the paper. There are some parameters about the model, and some parameters are about training. , such as epoch, etc., this kind of parameter has nothing to do with the model itself, in order to determine what parameters our TextCNN class needs to pass to initialize.

1. About models
   filter windows: [3,4,5]
   filter maps: 100 for each filter window
   dropout rate: 0.5
   l2 constraint: 3
   randomly select 10% of training data as dev set(early stopping)
   word2vec(google news) as initial input, dim = 300
   sentence of length: n, padding where necessary
   number of target classes
   dataset size
   vocabulary size
2. About training
   mini batch size: 50
   shuffuled mini batch
   Adadelta update rule: similar results to Adagrad but required fewer epochs
   Test method: standard train/test split ot CV

4.3 Dropout considerations

The solution is to solve the problem of fitting. When the last layer of softmax is a full-connected layer, it is easy to produce overfitting. The strategy is:

In the training phase, some dropouts are performed on the output of the max-pooling layer, activated with probability p, and the activated part is passed to the softmax layer.

In the test phase, w has been learned, but can not be used directly in unseen sentences, and is multiplied by p before reuse. This stage does not dropout all output to the softmax layer.

4.4 Input placeholder

```
1. # Placeholders for input, output and dropout
2. self.input_x = tf.placeholder(tf.int32, [None, sequence_length], na
   me="input_x")
3. self.input_y = tf.placeholder(tf.float32, [None, num_classes], name
   ="input_y")
4. self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_k
   eep_prob")
```

The tf.placeholder creates a placeholder variable. When training or testing, placeholders need to be entered into the network for calculations. The second parameter is the shape of the input tensor. None means it can be the length of any dimension, in our experiments it represents the size of the batch, and None allows the network to handle batches of any length.

The inactivation rate is also part of the input. Dropouts are used during training, and dropouts are not used during testing.

4.5 Embedding layer

This layer maps the word index to a low-dimensional vector representation. It is essentially a lookup table that we learn from data.

```
# Embedding layer
with tf.device('/cpu:0'), tf.name_scope("embedding"):
    W = tf.Variable(
        tf.random_uniform([vocab_size, embedding_size], 1.0, 1.0),
        name="W")
    self.embedded_chars = tf.nn.embedding_lookup(W, self.input_x)
    self.embedded_chars_expanded = tf.expand_dims(self.embedded_chars, -1)
```

Among them, W is the embedding matrix obtained during training, and is initialized with a random uniform distribution. Tf.nn.embedding_lookup embedding operation, get a 3-dimensional tensor, the shape is [None, sequence_length, embedding_size]. sequence_length is the length of the longest sentence in the data set, other sentences are added to this length by adding "PAD" . Embedding_size is the size of the word vector.

TensorFlow's convolution function, conv2d, takes four parameters, which are batch, width, height, and channel. Embedding does not include channel, so we artificially add it and set it to 1. Now it is [None, sequence_length, embedding_size, 1]

Stores a matrix of all word vectors <span tabindex="0" class="MathJax" id="MathJax-Element-1-Frame" role="presentation" style="display: inline-block; position: relative;" data- Mathml='W'>WW initialization is randomly random, which is the first model in the paper CNN-rand

Not every vocabulary is used in the training process, but only a batch is created (batch is a sentence, each sentence marks which word (maximum length is sequence_length), so the batch is equivalent to a two-dimensional list), This batch is input_x.

```
self.input_x = tf.placeholder(tf.int32, [None, sequence_length], name="input_x")
```

tf.nn.embedding_lookup: Finds all the ids in input_x and gets their word vector. Each sentence of each sentence in the batch must be searched. So the shape of embedded_chars should be

[None, sequence_length, embedding_size] (1)

However, after the input word vectors are obtained, the next step is to enter the convolutional layer using the tf.nn.conv2d function.

Take another look at the conv2d parameter list:

Input: [batch, in_height, in_width, in_channels] (2)

Filter: [filter_height, filter_width, in_channels, out_channels] (3)

Comparing (1)(2) can find that one in_channels is missing, and the most simple version has only 1 channel (Yon's fourth model uses multichannel)

Therefore, it is necessary to expand dim to adapt to the conv2d input requirements. The universal tensorflow has provided such functions:

```
1. This operation is useful if you want to add a batch dimension to a
   single element. For example, if you have a single image of shape [h
   eight, width, channels], you can make it a batch of 1 image with ex
   pand_dims(image, 0), which will make the shape [1, height, width, c
   hannels].
2. Example:
3. # ‘t’ is a tensor of shape [2]
4. shape(expand_dims(t, -1)) ==> [2, 1]
```

So only need

```
1. Tf.expand_dims(self.embedded_chars, -1)
```

You can add an in_channels=1 after embedded_chars

4.6 Conv and Max-pooling

We have filters of different sizes. Because each convolution produces a tensor of different shapes, we traverse each filter and then merge the results into one large feature vector.

```
1. # Create a convolution + maxpool layer for each filter size
2. pooled_outputs = []
3. for i, filter_size in enumerate(filter_sizes):
4.     with tf.name_scope("conv-maxpool-%s" % filter_size):
5.         # Convolution Layer
6.         filter_shape = [filter_size, embedding_size, 1, num_filters
   ]
```

```
            W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
            b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
            conv = tf.nn.conv2d(
                self.embedded_chars_expanded,
                W,
                strides=[1, 1, 1, 1],
                padding="VALID",
                name="conv")
            # Apply nonlinearity
            h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
            # Maxpooling over the outputs
            pooled = tf.nn.max_pool(
                h,
                ksize=[1, sequence_length - filter_size + 1, 1, 1],
                strides=[1, 1, 1, 1],
                padding='VALID',
                name="pool")
            pooled_outputs.append(pooled)

# Combine all the pooled features
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(3, pooled_outputs)
self.h_pool_flat = tf.reshape(self.h_pool, [1,num_filters_total])
```

Here W is the filter matrix and h is the result of a non-linear transformation of the convolution result. Each filter is stroked from the entire embedding, with the difference being how many words are covered. "VALID" padding means that the edge of the sentence is not padding, that is, using narrow convolution, the output shape is [1, sequence_length - filter_size + 1, 1, 1]. The difference between narrow convolution and wide convolution is whether to fill the edges. For example, for a sentence with 7 words, filter size is 5, use narrow convolution, the output size is (7-5) +1=3; using wide convolution, the output size is (7+2*4) -5)+1=11.

First, each filter_window_size in the filter_sizes must be convolved (each size must generate num_filters and then multiple filter maps), so the outer layer is a large for loop.

Since the filter_size is fixed inside the for loop, it can be combined with (3): [filter_height, filter_width, in_channels, out_channels], filter_shape = [filter_size, embedding_size, 1, num_filters].

The reason for clarifying the filter shape is to initialize the filter's weight matrix w:

```
1. W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name
   ="W")
```

The reason for using the tf.truncated_normal() function is:

Tensorflow provides two normal functions:

1. Tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)
2. Tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

By contrast, the parameter list of these two functions is exactly the same, and the difference is that I directly refer to the description in the document, and the explanation is very clear.

Outputs random values from a truncated normal distribution.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

In other words, the range of random values is within [mean-2 standard_deviations, mean + 2 standard_deviations].

The following figure can tell you where this range is,

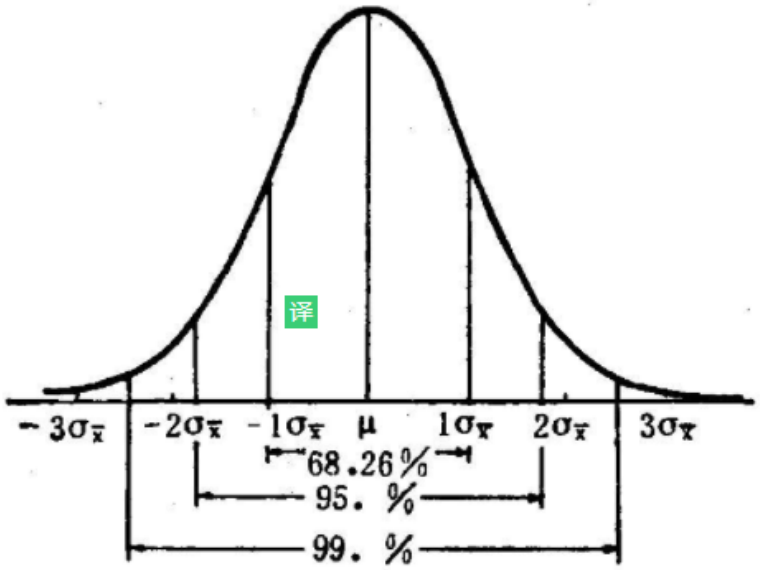


Figure 4.1 Range

After max-pooling the output, the tensor with the shape [batch_size, 1, 1, num_filters] is essentially a eigenvector, and the last dimension is the eigenvalue. Combine each tensor after max-pooling to get a long vector [batch_size, num_filters_total]. -1 in inf.reshape means that T flattens the vector.

4.7 Dropout

Dropout is perhaps the most popular regularization method in cnn. The idea of dropout is simple. The dropout layer randomly selects some neurons to inactivate them. This prevents co-adapting and forces them to learn useful features each. The number of inactive neurons is determined by dropout_keep_prob. Set 0.5 for training and 1 (disable dropout) for testing.

```
1. # Add dropout
2. with tf.name_scope("dropout"):
3.     self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep
   _prob)
```

Dropout only drops the output layer of hiddenlayer, so that some node values are not output to the softmax layer.

4.8 Scores and predictions

Using eigenvectors, we can use matrix multiplication to calculate the scores of the two types. We can also use the softmax function to calculate the two types of probability values.

```
1. with tf.name_scope("output"):
2.     W = tf.Variable(tf.truncated_normal([num_filters_total, num_cla
   sses], stddev=0.1), name="W")
3.     b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b"
   )
4.     self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")

5.     self.predictions = tf.argmax(self.scores, 1, name="predictions"
   )
```

4.9 Output layer

```
1. # Final (unnormalized) scores and predictions
2. with tf.name_scope("output"):
3.     W = tf.get_variable(
4.         "W",
5.         shape=[num_filters_total, num_classes],
6.         initializer=tf.contrib.layers.xavier_initializer())
7.     b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b"
   )
8.     l2_loss += tf.nn.l2_loss(W)
9.     l2_loss += tf.nn.l2_loss(b)
10.    self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")

11.    self.predictions = tf.argmax(self.scores, 1, name="predictions"
   )
```

The output layer is actually a softmax classifier.

4.10 Loss function

After getting the output of the entire network, that is, we get the y_prediction, but we still need to compare it with the real y label to determine the prediction.

The cross-entropy loss function used here

```
# CalculateMean cross-entropy loss
with tf.name_scope("loss"):
    losses = tf.nn.softmax_cross_entropy_with_logits(self.scores, self.input_y)
    self.loss = tf.reduce_mean(losses) + l2_reg_lambda * l2_loss
```

Or use the regular cross_entropy as a loss function. The last layer is a fully connected layer. To prevent overfitting, the l2 regularization term, l2_loss, is added to the loss func. L2_reg_lambda to determine the strength of the penalty.

4.11 Accuracy

```
# Accuracy
with tf.name_scope("accuracy"):
    correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, 1))
    self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, "float"), name="accuracy")
```

Tf.equal(x, y) returns a bool tensor, which is true if the xy counterparts are equal, otherwise false. The resulting tensor is [batch, 1].

Tf.cast(x, dtype) converts a bool tensor to a float type tensor for easy calculation Tf.reduce_mean() itself imports a vector of float type (elements are either 0.0 or 1.0). Calculating the mean directly on such a vector is accuracy. There is no need to specify reduction_indices.

4.12 Minimizing the loss

Optimize network loss with TensorFlow's built-in optimizers, such as Adam optimizer.

```
global_step = tf.Variable(0, name="global_step", trainable=False)
optimizer = tf.train.AdamOptimizer(1e-4)
grads_and_vars = optimizer.compute_gradients(cnn.loss)
train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)
```

Train_op is a new operation and we can perform gradient updates on the parameters. Each train_op is a training step. TensorFlow can automatically calculate those variables that are "trainable" and then calculate their gradients. The number of training steps can be calculated by the global_step variable, which is automatically incremented by one each time.

4.13 Checkpointing

Checkpointing can be used to save model parameters in TensorFlow. Checkpointing parameters can also be used to continue training.

```
# Checkpointing
checkpoint_dir = os.path.abspath(os.path.join(out_dir, "checkpoints"))
checkpoint_prefix = os.path.join(checkpoint_dir, "model")
# Tensorflow assumes this directory already exists so we need to create it
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
saver = tf.train.Saver(tf.all_variables())
```

4.14 Define a single training step

Use a batch of data for training.

```
def train_step(x_batch, y_batch):
    """
    A single training step
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
    }
    _, step, summaries, loss, accuracy = sess.run(
        [train_op, global_step, train_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step, loss, accuracy))
    train_summary_writer.add_summary(summaries, step)
```

Train_op doesn't return anything, just updates the parameters in the network. Finally, print out the current training loss and accuracy. If the size of the batch is small, the two differ greatly in different batches. Because dropouts are used, trained metrics may be worse than tested metrics.

The same function can also be used when testing.

```
def dev_step(x_batch, y_batch, writer=None):
    """
    Evaluates model on a dev set
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, summaries, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step,
    loss, accuracy))
    if writer:
        writer.add_summary(summaries, step)
```

4.15 Training loop

Train through iterative data.

```
def dev_step(x_batch, y_batch, writer=None):
    """
    Evaluates model on a dev set
    """
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, summaries, loss, accuracy = sess.run(
        [global_step, dev_summary_op, cnn.loss, cnn.accuracy],
        feed_dict)
    time_str = datetime.datetime.now().isoformat()
    print("{}: step {}, loss {:g}, acc {:g}".format(time_str, step,
    loss, accuracy))
    if writer:
```

```
16.         writer.add_summary(summaries, step)
```

Compare the accuracy of test and training sets.

Chapter 5
Experiment

5.1 Datasets

The data set used for the experiment is as follows (specific names and sources can refer to the paper)

Table 5.1 Datasets

Data	c	l	N	$\|V\|$	$\|V_{pre}\|$	*Test*
MR	2	20	10662	18765	16448	CV
SST-1	5	18	11855	17836	16262	2210
SST-2	2	19	9613	16185	14838	1821
Subj	2	23	10000	21323	17913	CV
TREC	6	10	5952	9592	9125	500
CR	2	19	3775	5340	5046	CV
MPQA	2	3	10606	6246	6083	CV

In this experiment, all aspects of the MR database were tested.

Movie Review Data is a distribution site for movie-review data for use in sentiment-analysis experiments.

5.2 Environment for experiment

In this experiment, a remote server mobaxterm was used to connect with a lab computer to conduct experiments.

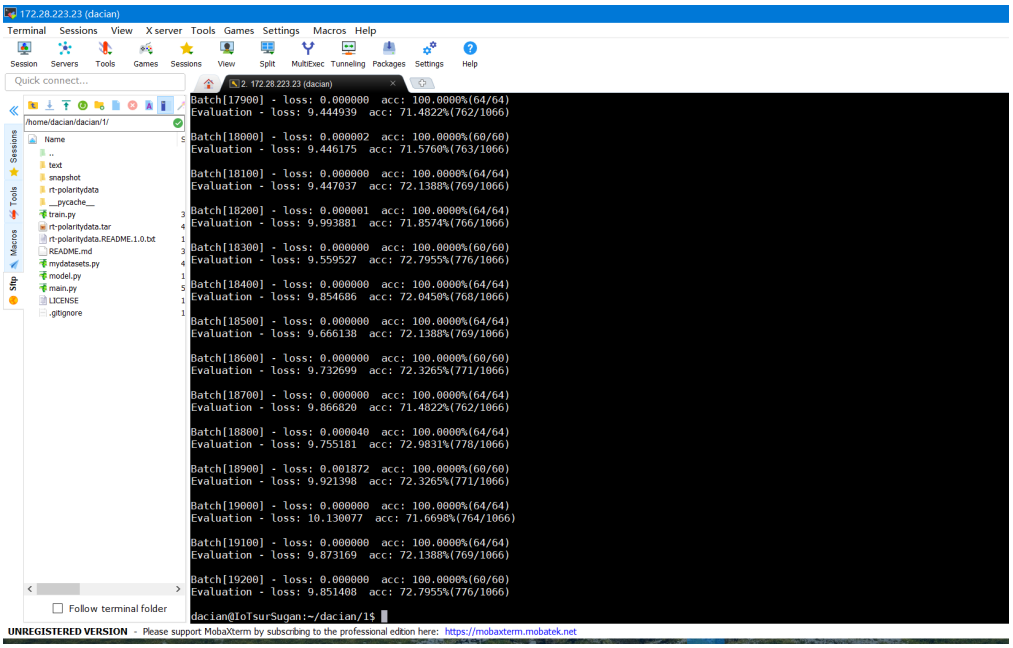


Figure 5.1 Remote server mobaxterm

5.3 Model training and adjustment

Corrected linear units

The h-size of the filter: 3, 4, 5; the number of corresponding Feature Maps is 100;

Dropout rate is 0.5, L2 regularization limit weight size does not exceed 3;

The size of mini-batch is 50;

The selection of these parameters is based on the MR dataset and the optimal parameters obtained by the Grid Search method (Grid Search). In addition, the stochastic gradient descent method was used during training.

The table below is part of the tuning process：

Table 5.2 Tuning process

lr	batchsize	epoch	embedding dimension	pool	loss	acc
0.001	64	256	128	max	10.3	73%
0.005	64	256	128	max	277	74%
0.0005	64	256	128	max	5.5	73%
0.001	256	256	128	max	5.3	70%

0.001	128	256	128	max	8	71%
0.001	64	256	128	max	12	71%
0.001	64	256	128	max	12	70%
0.001	64	512	128	max	24	72%
0.001	64	256	256	max	25	75%
0.001	64	128	256	max	8.2	73%
0.001	64	128	256	avg	5.5	73%

5.4 Pre-trained Word Vector

Word2vec is also called word embeddings, the Chinese name "word vector". The role is to turn the words in natural language into Dense Vectors that the computer can understand. Prior to word2vec, natural language processing often turned words into discrete individual symbols, namely One-Hot Encoder.

5.5 Training process

My results on the remote server are as follows:

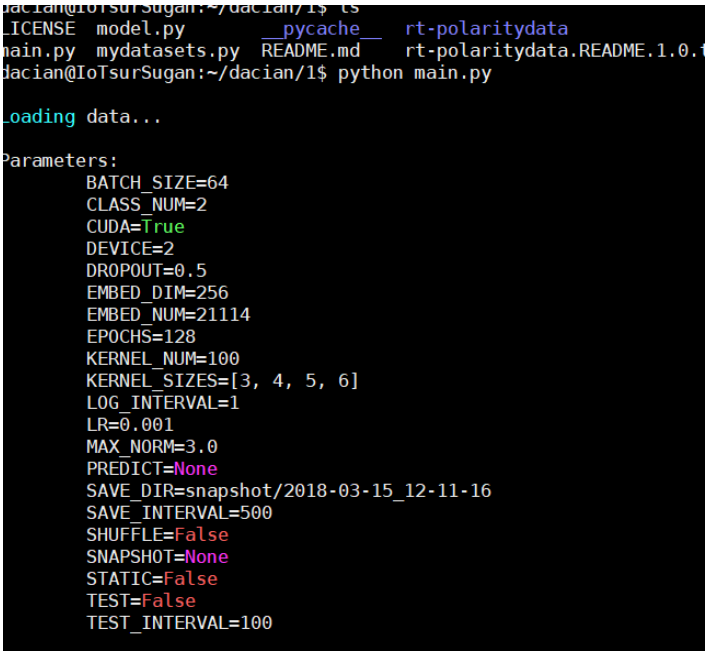


Figure 5.2 Training process 1

```
Batch[18800] - loss: 0.000040  acc: 100.0000%(64/64)
Evaluation - loss: 9.755181  acc: 72.9831%(778/1066)

Batch[18900] - loss: 0.001872  acc: 100.0000%(60/60)
Evaluation - loss: 9.921398  acc: 72.3265%(771/1066)

Batch[19000] - loss: 0.000000  acc: 100.0000%(64/64)
Evaluation - loss: 10.130077  acc: 71.6698%(764/1066)

Batch[19100] - loss: 0.000000  acc: 100.0000%(64/64)
Evaluation - loss: 9.873169  acc: 72.1388%(769/1066)

Batch[19200] - loss: 0.000000  acc: 100.0000%(60/60)
Evaluation - loss: 9.851408  acc: 72.7955%(776/1066)

dacian@IoTsurSugan:~/dacian/1$
```

Figure 5.3 Training process 2

5.6 Very Deep CNN architecture

For comparison, the advanced framework VDCNN was used for experiments. Using the following structure, three types of data such as ag_news、db_pedia and yelp_review were applied and the results were obtained.

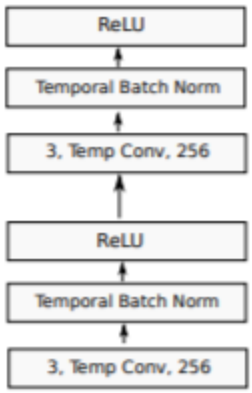


Figure 5.4 convolutional block

Table 5.4 Data sets for VDCNN

Dataset	Classes	Train samples	Test samples
AG's News	4	120 000	7 600
DBPedia	14	560 000	70 000
Yelp Review Full	5	650 000	50 000

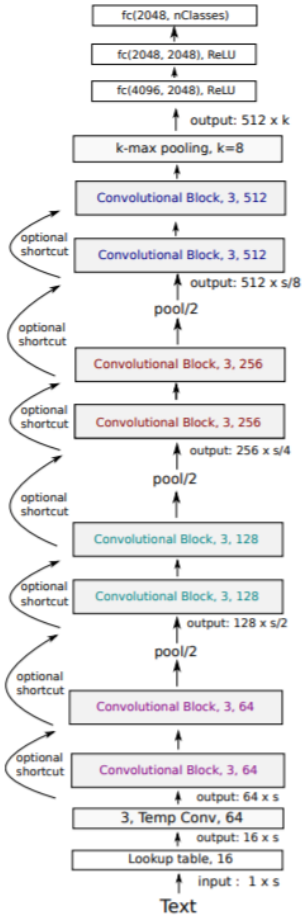


Figure 5.5 VDCNN architecture

Training process is as followed.

```
dacian@IoTsurSugan:~/dacian/final$ python3 train.py -d dbpedia
Training: Iteration: 1/4375 Epoch: 1/20 Loss: 20.15843605156988 Accuracy: 0.0625
Training: Iteration: 2/4375 Epoch: 1/20 Loss: 18.378523744642735 Accuracy: 0.1171875
Training: Iteration: 3/4375 Epoch: 1/20 Loss: 17.40357880666852 Accuracy: 0.0625
Training: Iteration: 4/4375 Epoch: 1/20 Loss: 8.660828160122037 Accuracy: 0.078125
Training: Iteration: 5/4375 Epoch: 1/20 Loss: 14.347839390859008 Accuracy: 0.1015625
Training: Iteration: 6/4375 Epoch: 1/20 Loss: 9.605724451132119 Accuracy: 0.078125
```

Figure 5.6 Training process

I almost keep the default settings as described in the article. For the optimizer, I use the Adam optimizer and the initial learning rate is 0.001 instead of 0.01.

The command for running this code is : python train.py -d dataset_name, for instance: python train.py -d dbpedia

Table 5.5 Comparison with results in paper

Data set	Accuracy of experiment	Accuracy in paper
Ag_news	89.0625	90.14
DB_Pedia	98.439	98.44
Yelp_review	67.18	61.96

Chapter 6
Conclusion

6.1 Experience

Understand data: Although the application of deep learning has a big advantage is that it no longer requires cumbersome and inefficient manual engineering, but if you just think of him as a black box, it is inevitable that people will always doubt life. Be sure to understand data and remember that data sense is always important, regardless of traditional methods or deep learning methods. Pay attention to the analysis and understand why your data is suitable and why.

Focus on the quality of iterations - Record and analyze each of your experiments: Iteration speed is the key to determining the success or failure of an algorithm project, and students who learn the probability are easily identified. The important thing of the algorithm project is not only the iteration speed, but also the quality of iterations. If you don't build a quick experimental analysis routine, then iterating faster will only be a bad resource for your company. It is advisable to record each experiment. The experimental analysis answers at least these three questions: Why experiment? What is the conclusion? How to experiment next?

Hyperparametric adjustment: Hyperparametric adjustment is the daily routine for all engineers. We recommend a text classification practice paper (A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification, which contains a comparison of some super parameters. Experiments, if you are just beginning to start a text analysis task, you may wish to set the parameters by the result of the article. How to get the maximum parameter adjustment is actually a very important issue. You can read this essay on bleating. - Know that column.

Must use dropout: There are two things you can do without: the amount of data is particularly small, or you use a better regular method, such as bn. In practice, we tried dropouts with different parameters, the best being 0.5, so if your computing resources are very limited, the default 0.5 is a good choice.

Fine-tuning is a must-have: I've talked to you before. If you just use word2vec-trained word vectors as feature representations, I bet you will lose a lot of results.

Softmax loss does not necessarily have to be: depending on your data, if your task is non-mutually exclusive among multiple classes, you can try to train multiple classifiers, and our adjusted accuracy is still increased by >1%.

Unbalanced category issues: Basic is a conclusion that has been verified in many scenarios: If your loss is dominated by a subset of categories, it is mostly negative for the overall situation. It is recommended to try a similar booststrap method to adjust the sample weight in the loss.

Avoid training churn: By default, random sampling factors must be added to make the data distribution iid as much as possible. The default shuffle mechanism can make the training result more stable. If the training model is still turbulent, consider adjusting the learning rate or mini_batch_size.

6.2 Improvement

We can see from the result of experience:

Due to random factors in the training process of the model, such as the weight parameters initialized randomly, mini-batch, and stochastic gradient descent optimization algorithms, the results of the model on the data set have a certain degree of fluctuation, for example, the accuracy can reach 1.5. % of float, while AUC is 3.4%.

Whether the word vector uses word2vec or GloVe has a certain influence on the experimental results. Which one is better depends on the task itself.

The size of the Filter has a large impact on the performance of the model, and the parameters of the Filter should be updatable;

The number of Feature Map also has a certain influence, but it needs to take into account the training efficiency of the model;

1-max pooling is good enough, compared to other pooling methods;

The role of regularization is minimal.

6.3 Development

How will NLP develop in the next five to ten years?

1) The progress of question and answer and reading comprehension will make the search engine more accurate;

2) Speech recognition and neural machine translation will make spoken machine translation completely practical;

3) Promoting information services and advertising is more natural, friendly and personalized due to the accuracy and real-time improvement of user portraits;

4) Improve chat, question and answer and dialogue techniques, and promote natural language conversations to be practical;

5) Due to the advancement of dialog technology and knowledge maps, intelligent customer service and manual customer service are more perfectly combined, which greatly improves customer service efficiency;

6) Because of the progress of natural language generation technology, automatic writing of poems, composing music, automatically generating news and even novels will become popular;

7) The progress of human-machine dialogue has promoted the popularity of voice assistants, Internet of Things, smart hardware, and smart homes;

8) Finally, NLP+ means that NLP is widely used in vertical fields such as finance, law, education, and medical.

Take the search engine intelligence as an example. In previous search engines, entering keywords returned a bunch of things that you needed to see for yourself. With the improvement of automatic question and answer, reading comprehension and other capabilities, the current search engine, you can ask questions, the sentence is not afraid of long, it can analyze the question, find out the answers from the vast number of documents; not only to You have a link to a document, it can also give you an answer directly, and the search engine results are becoming more and more accurate.

Future directions for NLP research need to be focused on:

1) Personalize services through user portraits;

2) Insight into the artificial intelligence mechanism through interpretable learning;

3) Improve learning efficiency through the combination of knowledge and deep learning;

4) Realize domain adaptation through migration learning;

5) Continuous evolution through reinforcement learning;

6) Make full use of unlabeled data through unsupervised learning;

7) Understanding, questioning, and conversion between multimedia and multimodality.

From the opposition of symbolism and connectionism to cooperation, from static analysis to interaction, from grammar and shallow semantics to deep semantics, from functionalism to cognitive and emotional experience.

References

[1] Yoshua Bengio, R´ejean Ducharme, Pascal Vincent, and Christian Janvin, A neural probabilistic language model. The Journal of Machine Learning Research, 3:1137–1155, 2003.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, Distributed Representations of Words and Phrases and their Compositionality. NIPS'13 Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 Pages 3111-3119

[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov, Bag of Tricks for Efficient Text Classification. arXiv:1607.01759v3 [cs.CL] 9 Aug 2016

[4] Fukushima K. Analysis of the process of visual pattern recognition by the neocognitron[J]. Neural Networks,1989, 2(6):413 - 420.

[5] Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition[J]. Neural Networks, 1988, 1(2):119 - 130.

[6] Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner. Gradient Based Learning Applied to Document Recognition. PROC OF THE IEEE NOVEMBER 1998

[7] Denny Britz, Implementing a CNN for Text Classification in TensorFlow http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/

[8] Kim, Yoon, Convolutional Neural Networks for Sentence Classification, eprint arXiv:1408.5882 08/2014

[9] Ye Zhang, Byron C. Wallace, A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification, arXiv:1510.03820v4 [cs.CL] 6 Apr 2016

[10] Shawn1993, cnn-text-classification-pytorch, https://github.com/Shawn1993/cnn-text-classification-pytorch