A useful hanbook for commonly-used neural networks for deep learning(MLP, RNN, CNN..)

Chan Li[a]

[a]Department of Physics, Sun Yat-sen University

August 23, 2022

Contents

1 Introduction

In this handbook, I will present some commonly-used (though very old style) architectures of deep learning for new beginners, including multiple layer perceptron(MLP), Convolutional Neural Network (CNN), Recurrent neural networks (RNN) and Hopfield network (which is actually a type of recurrent one). In each section, I will represent the mathematical form of a given network architecture, including the dynamics and the derivation of updating functions. Full set of code is given in the github website, given by pure python (all you need is python!), perfect for a better understanding of the intrinsic mechanisms. Now, let's begin the adventure!

2 Commonly used datasets

In order to keep the relative consistence among all the network architectures, I introduce three commonly-used datasets: MNIST dataset, Fashion-MNIST dataset and CIFAR-10 dataset, which will be utilized in training the following networks. Typically, these datasets are used to implement classification tasks.

MNIST dataset, four ingredents are included: 60000 pictures and corresponding labels in training set, 10000 pictures and corresponding labels in testing set. The range of pixels is often scaled between (0,1). The examples are shown below in Fig.1. If you want to download this dataset in your code and utilize this, please see MNIST.

CIFAR 10 and CIFAR 100 datasets contains natural images which are much more difficult to classify, including bird, cat, deer... and so on, shown in Fig. 2. In this dataset, there are 50000 training pictures along with training labels, and 10000 test pictures with corresponding labels.

There are also other datasets, like Fashion-MNIST. But for simplicity, we use here only the MNIST and CIFAR-10 datasets.

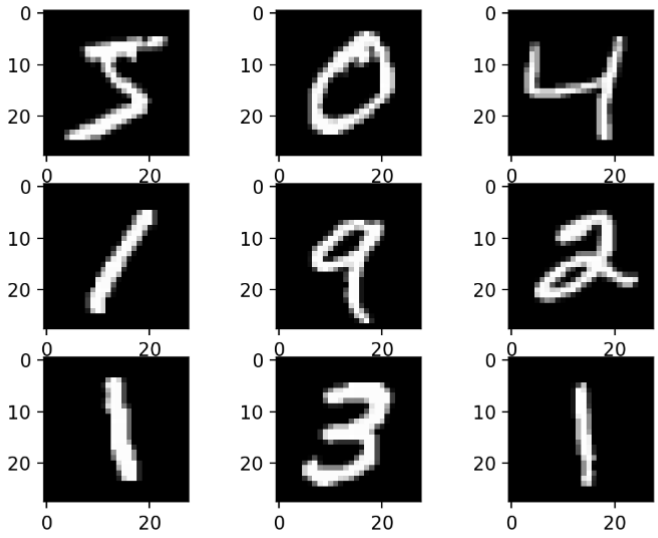

Figure 1: Examples of MNIST pictures (copy from the Internet)

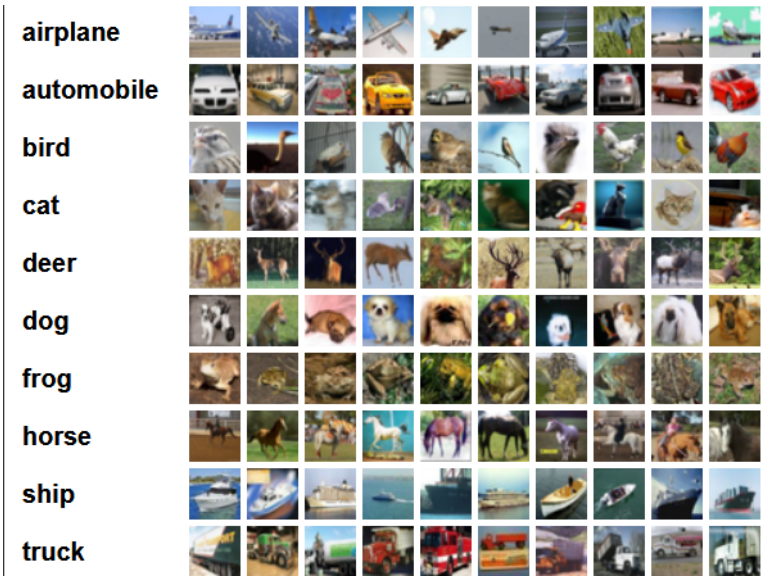

Figure 2: Examples of CIFAR10 pictures (copy from the Internet)

3 Commonly used architectures – Supervised learning

Supervised learning: We get data and corresponding labels beforehand, which is the main focus of this document.

3.1 MLP(multi-layer-perceptron)

In this section, I will introduce a commonly-used framework for deep learning: multi-layer-perceptron. The illustration of this architecture is shown in Fig.3.

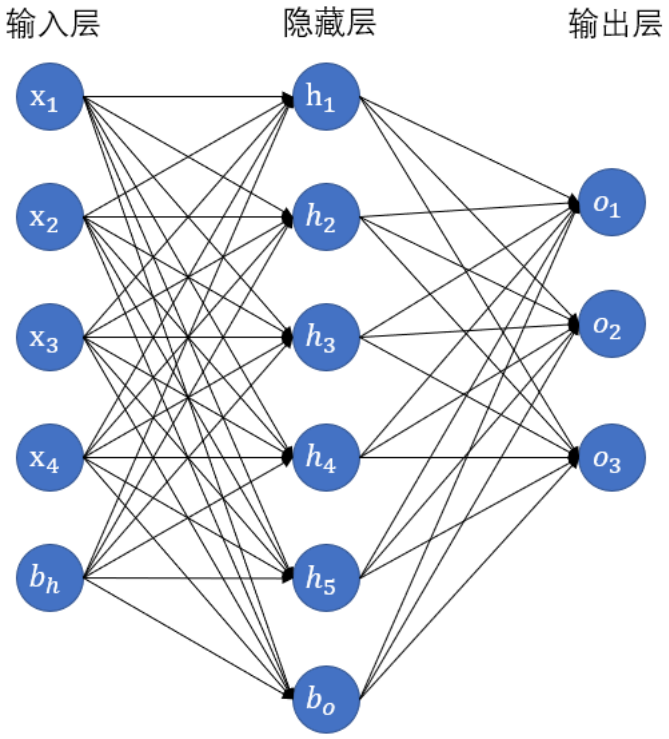

Figure 3: Multi-layer-perceptron (picture copied from internet).

First define the input data $x \in \mathbb{R}^{N\times M}$,N denotes the number of input neurons, while M indicates the number of data samples fed into the network. $l = 0, ..., L$ denotes the index of layers, w_{ij}^l characterizes the interaction of neuron i in layer l and neuron j in layer $l-1$. For each neuron i layer l, there exist pre-activation z_i^l

and activation h_i^l illustrated below:

$$\begin{aligned} z_i^l &= \sum_{ij} w_{ij}^l h_j^l, \\ h_i^l &= f(z_i^l). \end{aligned} \tag{1}$$

The function $f(\cdot)$ is the activation function, often chosen to be sigmoid function or **ReLU** function(which is more biologically-plausible). In the output layer, **softmax** is a good choice to specify the probability over all the classes.

$$\mathbf{softmax}(z_i^l) = \frac{e^{z_i^L}}{\sum_j e^{z_j^L}}. \tag{2}$$

The goal of this network is to classify the pictures to their labels. For the MNIST dataset, ten classifications are considered: $0, 1, ..., 9$, and we often use one-hot coding for the labels. For example, for picture with label 3, we use $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. To achieve the classification goal, we try to minimize the loss function, which is chosen dependent on the characteristics of the task. For classification tasks, usually the cross-entropy loss function is chosen, which is defined of the distribution q relative to a distribution p over a given set:

$$H(p, q) = -\mathrm{E}_p[\log q], \tag{3}$$

where in classification tasks, we can rewrite it for a specific input sample as:

$$\mathcal{C} = -\sum_i \hat{y}_i \ln(h_i^L), \tag{4}$$

where $\hat{\mathbf{y}}$ denotes the corresponding label, while $\mathbf{h}^L$ shows the network output and the value of cross entropy is averaged over all samples in a dataset. It is clearly seen that, if the network has classified the images perfectly, the value is close to zero, or it grows rather large ($\langle \mathcal{C} \rangle = 2.6$ for chance level). Next, we derive the

updating function for parameter $\boldsymbol{w}$. To begin with,

$$\begin{aligned}\Delta w_{ij}^l &= -\eta \frac{\partial \mathcal{C}}{\partial z_i^l} \frac{\partial z_i}{\partial w_{ij}}, \\ &= -\eta \mathcal{K}_i^l \frac{\partial z_i^l}{\partial w_{ij}},\end{aligned} \tag{5}$$

where η denotes the learning rate and we define $\mathcal{K}_i^l$ as the error signal, which could be obtained using chain rule. Generally, $\mathcal{K}_i^L = \frac{\partial \mathcal{C}}{\partial z_i^L}$ has direct form, based on the choice of loss function and transfer function, i.e., the output transfer function is **softmax** and cross entropy is applied, $\mathcal{K}_i^L = h_i^L - \hat{y}_i$. Based on the error signal from the output layer:

$$\begin{aligned}\mathcal{K}_i^l &= \sum_j \frac{\mathcal{L}}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l}, \\ &= \sum_j \mathcal{K}_j^{l+1} w_{ji}^l f'(z_i^l),\end{aligned} \tag{6}$$

where $f'(\cdot)$ denotes the derivative of transfer function $f(\cdot)$. Taken together,

$$\begin{aligned}\Delta w_{ij}^l &= -\eta \frac{\partial \mathcal{C}}{\partial z_i^l} \frac{\partial z_i}{\partial w_{ij}}, \\ &= -\eta \mathcal{K}_i^l h_j^l, \\ w_{ij}^l &= w_{ij}^l + \Delta w_{ij}^l.\end{aligned} \tag{7}$$

By iteration, the value of $\mathbf{w}$ and cross entropy converges and we get a well-trained MLP. Some tips for training: initialization is often chosen to be HeKaiming type where $w_{ij} \sim \mathcal{N}(0, \frac{1}{N_{l-1}})$, learning rate can be fixed at $\eta = 0.01$ or decay with training stages. Here is an example showing the test accuracy (validated on $V = 10000$ unseen data) of network structure $[784, N, N, 10]$.

3.2 CNN (Convolutional Neural Networks)

CNN is a typical example of biologically-inspired networks, which has similar structure as human ritina, therefore is often used in visual imagery. In this section, I will introduce two types of CNN, the classical Lenet for MNIST and another architecture handling CIFAR-10 dataset.

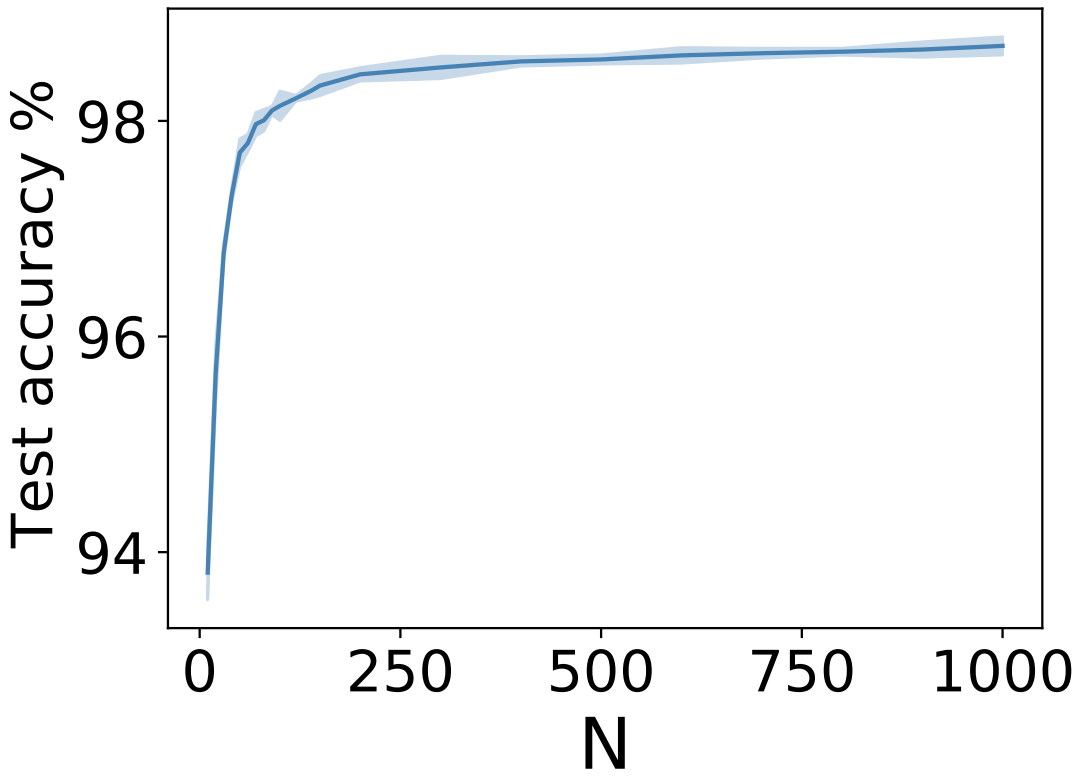

Figure 4: The test accuracy of network structure $[784, N, N, 10]$, the fluctuation is averaged over five independent runs.

3.2.1 Convolution

Convolution in deep learning is different from convolution in mathematics or image processing. Convolution in deep learning (strictly speaking, cross-correlation) is that convolution kernel traverses the original image, corresponding elements are multiplied and summed, and the size of the new image obtained will be reduced. In fully connected neural networks, image data and features are stored in the form of column vectors. In convolutional neural networks, the data format is mainly stored in the form of tensor (which can be understood as multi-dimensional array). The format of the picture is a three-dimensional tensor, line × column × channels. The format of convolution kernel is a four-dimensional tensor, and the number of convolution kernels ×line × column × channels.

The convolution operation is to extract one convolution kernel at a time. The format of one convolution kernel is three-dimensional and is line × column × channels, and we define line or column as the $filtersize$ of the kernel. After the two-dimensional convolution operation of the picture corresponding to the channel

number and the convolution kernel, the convolution result corresponding to the channel is obtained, and the results of all channels are added to obtain one channel of the output image. Each convolution kernel corresponds to one channel of the output image, that is, the number of channels of the output image is equal to the number of convolution kernels.

First, confirm the dimension of the input image: row ×column ×channels, where the number of channels is related to the color of the image.

$$\mathbf{h}^1 = f(\mathbf{z}^1) = f(\mathbf{W}^0 * \mathbf{0} + \mathbf{b}^1), \tag{8}$$

where the superscript represents the index of layers, $\mathbf{h}$ denotes the activation value tensor, $\mathbf{v}$ represents the input picture (which is a tensor), the asterisk $*$ represents the convolution operation, $\mathbf{b}$ is the network bias, and the connection $\mathbf{W}$ is also a tensor. Here, the CNN model parameters we need to define ourselves are:

1) In general, we have more than one convolution kernel, for example K, therefore there are K channels for the output of our input layer, or the corresponding input of the second convolution layer.

2) For the size of each submatrix in the convolution kernel, we generally use the submatrix as the convolution kernel of the square matrix, such as the submatrix of $F \times F$.

3) Padding (hereinafter referred to as P): when we convolute, in order to better identify the edges, we usually add several circles of 0 around the input matrix and then convolute. The number of circles will be the number of P.

4) Step stride (hereinafter referred to as s), i.e. the distance of each pixel moved in the convolution process.

Assuming that the original input image is $m \times m$, the output image is $n \times n$, the number of zero padding is p, the convolution kernel is $f \times f$, and the convolution kernel sliding step is s, the output size is

$$n = (m + 2p - f)/s + 1 \tag{9}$$

3.2.2 Pooling

The so-called pooling is to downsample the picture. The maximum pooling is to represent the region with the maximum value of each region in the picture, and the average pooling is to represent the region with the average value of each region. Assuming that the output of the convolution layer is $n \times n$, and the pooling matrix has the size of $K \times K$, then the size after pooling is $\frac{n}{K} \times \frac{n}{K}$. We need to decide:

1) Size of pooling area k.

2) The standard of pooling is generally Max or average.

3.2.3 Fully-connected layers

The fully-connected layers takes the form of common structure of MLP, so we can directly use the forward propagation of it:

$$\mathbf{h}^l = f(\mathbf{z}^l) = f(\mathbf{W}^{l-1}\mathbf{h}^{l-1} + \mathbf{b}^l) \tag{10}$$

After several fully-connected layers, the last layer is the **softmax** output layer. The only difference between the output layer and the normal fully-connected layers is the activation function. Here, we need to decide:

1) Activation function of hidden layers.

2) Number of neurons in each layer.

3.2.4 Updating parameters

In this subsection, we will discuss the updating of parameters for all the layers. As we have mentioned before:

$$\begin{aligned} \Delta w_{ij}^l &= -\eta \frac{\partial \mathcal{C}}{\partial z_i^l} \frac{\partial z_i}{\partial w_{ij}}, \\ &= -\eta \mathcal{K}_i^l h_j^l, \\ w_{ij}^l &= w_{ij}^l + \Delta w_{ij}^l. \end{aligned} \tag{11}$$

The error signal $\mathcal{K}^l$ can be derived using chain rule. For the fully-connected layers, we can easily get it from the previous section about MLP:

$$\begin{aligned} \mathcal{K}_i^l &= \sum_j \frac{\mathcal{L}}{\partial z_j^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_i^l}, \\ &= \sum_j \mathcal{K}_j^{l+1} w_{ji}^l f'(z_i^l). \end{aligned} \tag{12}$$

Considering the error signal δ^l backpropogated to the pooling layer, we can derive the form of δ^{l-1} as:

$$\delta_k^{l-1} = \left(\frac{\partial h_k^{l-1}}{\partial z_k^{l-1}}\right)^T \frac{\partial \mathcal{L}(W,b)}{\partial h_k^{l-1}} = \text{ upsample } \left(\delta_k^l\right) \odot \sigma'\left(z_k^{l-1}\right),$$

$$\boldsymbol{\delta}^{l-1} = \text{ upsample } \left(\boldsymbol{\delta}^l\right) \odot \sigma'\left(\boldsymbol{z}^{l-1}\right),$$

where the upsample is an operation based on the type of pooling method. There are three cases: maximum pooling, average pooling and global pooling. Then, if it is Max (max pooling is most commonly used), we first reconstruct a zero matrix with the same size as the activation before pooling, and place δ_i^l to the position with the largest activation value.

Next, if the error signal δ^l is backpropogated to the convolution layer, we can derive it as:

$$\delta^{l-1} = \left(\frac{\partial z^l}{\partial z^{l-1}}\right)^T \delta^l = \delta^l * \text{rot}\,180\left(W^l\right) \odot \sigma'\left(z^{l-1}\right),$$

where rot180$(\cdot)$ denotes flip up and down once, then flip left and right once, the derivation is easy if we take a simple example to see. Now, we have obtained the form of δ^l in all cases and layers, updating the parameters is based on δ^l. For the fully-connected layers:

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^l} = \delta_i^{l+1} \times h_j^l. \tag{13}$$

While there is no need for updating parameters in pooling layers. In the case of

convolution layers,

$$\begin{aligned} \frac{\partial \mathcal{L}(W,b)}{\partial W^l} &= h^l * \delta^{l+1}, \\ \frac{\partial \mathcal{L}(W,b)}{\partial b^l} &= \sum_{u,v} \left(\delta^l\right)_{u,v}, \end{aligned} \tag{14}$$

where the gradient of bias is based on the dimension of it. So far, we have derived all the details about updating parameters **W** and **b** in CNN.

3.2.5 A classical architecture: Lenet

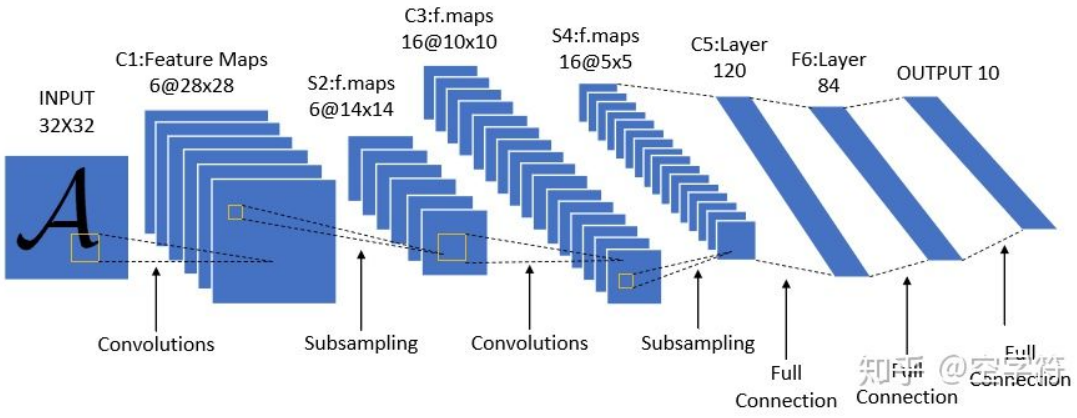

Figure 5: The illustration of the Lenet.

1. If the input picture is $28 \times 28 \times 1$ and padding is 2, the input picture will eventually have the size of $32 \times 32 \times 1$. The first convolution layer can be called C_1, the input is a picture of $32 \times 32 \times 1$, the convolution kernel size is 5×5, 6 kernels are considered, the strip is 1 pixel, finally the output size is $28 \times 28 \times 6$. The trainable parameters are $(5 \times 5 + 1) \times 6 = 156$. Number of neurons in this layer: $28 \times 28 \times 6$. Since the forward is carried out by convolution, although the number of weights is very large, we do not need to update these weights, but only need to update the convolution kernel. So how do we understand the weight sharing of convolution layers? The so-called weight sharing means that for an input image, a filter is used to scan the image, and the number in the filter is the weight. Each position in the picture is scanned by the same filter, so the weight is the same, which is sharing. This can greatly reduce the number of weights that need to be trained.

2, followed by s_2 pool layer. The input size is 28×28, and the pooling size is 2×2. Therefore, the size of the feature map is $14 \times 14 \times 6$. For convenience, the

method of maximum pooling can be used here, so the trainable parameter here is 0.

3. The third layer is another convolution layer C_3. The size of convolution kernel is 5×5, and the number of convolution kernel is 16. After the pooling layer, how to calculate the size of the feature map?

$$output = (input + 2 \times p - kernel)/stride + 1 = ((14 + 2 \times 0 - 5)/1) + 1 = 10 \quad (15)$$

The first 6 feature maps in C_2 are generated by the first three adjacent output

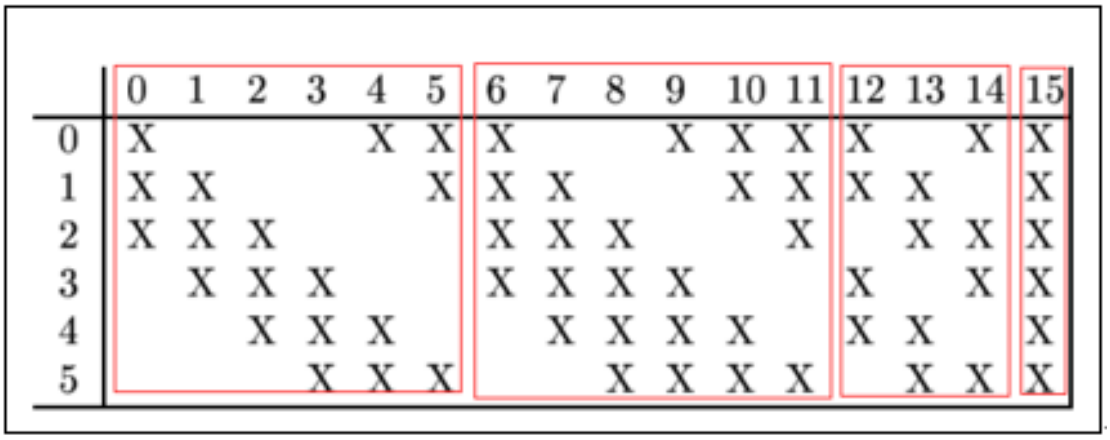

Figure 6: Generating feature maps in C_3 with different combination of maps from S_2.

maps in S_2, followed by the next 6 feature maps correlated with the first four adjacent output maps in S_2, and so on, as shown in Fig.6. Note that in this process, each combination can be assigned a convolution kernel, so that a total of 16 combinations are matched with 16 convolution kernels. There are two reasons for this: reducing parameters; This asymmetric combination is helpful to extract multiple combination features.

4. The fourth layer is a pooling layer. The input size is 10×10, the pooling area is 2×2, and the sampling method is: add Max pooling.

5, A convolution layer is attached next, the input comes from S_4, 16 feature maps in total. The size of kernel: 5×5, the number of kernels: 120, 120 feature maps are output with the size 1×1:

$$output = (input + 2 \times padding - kernel)/stride + 1 = ((5 + 2 \times 0 - 5)/1) + 1 = 1 \quad (16)$$

6, Followed by the F_6 fully-connected layer, F_6 and C_5 are connected. Input size: the output of layer C_5 with the size (120×1), output size 84×1 as the hidden layer has 84 neurons.

7, Finally, the output layer has ten nodes which equals the total number of classes, and we use **softmax** function to specify probability distribution over ten classes.

Without doubt, on MNIST dataset, we can reach test accuracy about 99.3%.

3.2.6 A architecture to copy with CIFAR 10 dataset.

In this mode, the network includes 5 layers in total, in which 4 convolution layers are considered, with the layer width $[k, 2k, 4k, 8k]$ and we choose $k = 64$ in most cases, finally a fully-connected layer is used to output the results. For all the convolution layers, the size of kernels is 3×3, stride $= 1$ and padding=1, followed by the pooling layer with the size of $[1, 2, 2, 8]$.

Next, we will analyze the parameters of this model in detail. First, the size of the input picture is $32 \times 32 \times 3$. After the first convolution layer, the size is $K \times 3 \times 3 \times 3$, and the output size is $32 \times 32 \times 6$. Next, the batch normalization is implemented, and finally the relu non-linear is used. Followed by the pooling layer, the pooling window is 1×1, then the output feature map size is still $32 \times 32 \times K$. Next, enter the next convolution layer. The size of the convolution layer is $2 \times K \times 3 \times 3 \times K$. After batch normalization and relu nonlinearity, the output feature map is $32 \times 32 \times 2K$. After the pooling with area 2×2, the output is $16 \times 16 \times 2K$. After the third convolution layer, the filter size is $4 \times K \times 3 \times 3 \times 2K$, after batch normalization, the nonlinear output size is $16 \times 16 \times 4K$, and after the pooling with size 2×2, the output size is $8 \times 8 \times 4K$. Entering the fourth convolution layer, the convolution kernel size is $8K \times 3 \times 3 \times 4K$, and after batch normalization and nonlinear output, the size is $8 \times 8 \times 8K$. After the last layer of pooling, the output of $1 \times 1 \times 8K$ is obtained, and finally connected with the output of 10 neurons.

Using this model, on CIFAR-10 dataset, we can reach test accuracy at about 85%.