HOMEWORK 5

Jiazhi Li

jiazhil@usc.edu

April 4, 2019

1 CNN Architecture and Training

1.1 Abstract and Motivation

Convolutional Neural Networks (CNN) is a type of feed-forward neural network with convolutional computation and deep structure. It is one of the representative algorithms of deep learning. The study of convolutional neural networks began in the 1980s and 1990s. Time Delay network and LeNet-5 were the earliest convolutional neural networks. After the 21st century, with the introduction of deep learning theory and the improvement of numerical computing equipment, convolutional neural networks have been rapidly developed and widely used in computer vision and natural language processing [1].
The convolutional neural network constructs the visual perception mechanism of the creature, which can perform supervised learning and unsupervised learning. The convolutional kernel parameter sharing and the sparseness of the inter-layer connection in the hidden layer enable the convolutional neural network to reduce the number of computational weights for grid-like topology features such as pixel and audio, which have a stable effect and no additional feature engineering requirements for the data.

1.2 Discussion

1.2.1 CNN components

Just like the regular ANN, the CNN is also consisted with several layers. The layers are followed by each other, which is like a sequence of layers. The three main types of CNN are Convolutional Layer, Pooling Layer and Fully-Connected Layer. Convolutional Layer and Pooling Layer are working for feature extraction. Fully-Connected Layer is used for classification. What's more, between each two Layer, kind of differentiable activation function are used to transform. This is the basic structure of CNN. The architecture of LeNet-5 is shown in **Figure** 1. This structure is consisted with two Convolutional layers followed by two Pooling Layers and three Fully-Connected Layers. The more details about each layer and the function are introduced as followed.

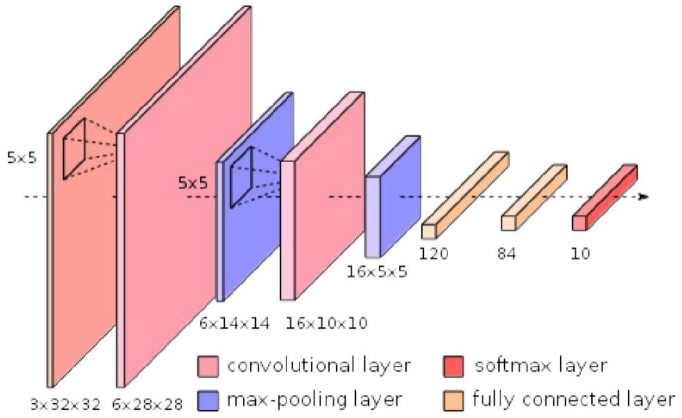


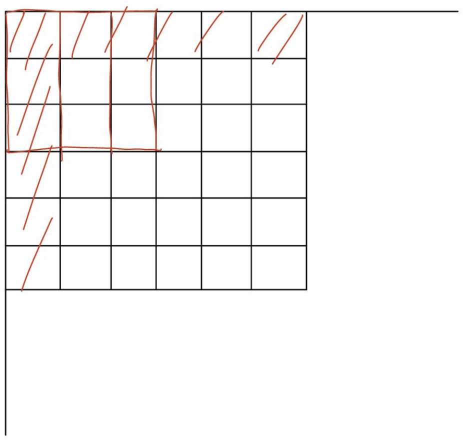
Figure 1: CNN architecture of LeNet-5

`Input Layer`

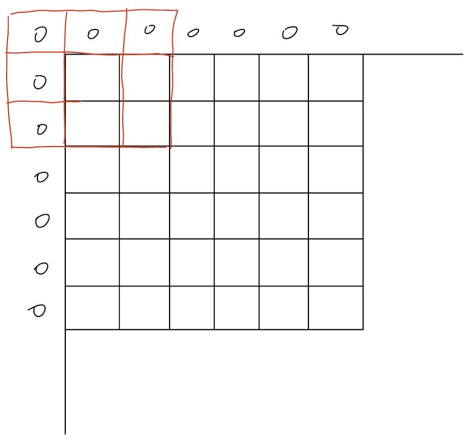
The first layer is input layer which provides the input pixel to the network. The size of this layer depends on the size of training images and testing images. For the LeNet-5 on MNIST Dataset, the size is 32*32*3 in the image of width 32, height 32, and with three color channels R,G,B so that there will be 32*32*3 neurons.

`Convolutional Layer`

In this layer, what the CNN do is feature extraction. In traditional regular ANN, there is no convolutional layer. Instead, the full connectivity layer is used as all hidden layers. If we connect all input neurons with next layer just like what we do in fully connected layer, there will be 32*32*3 = 3072 weights. To be mentioned, this is just the first layer. After the next several layers, the parameters would add up quickly. If there are no sufficient training samples, it would cause over-fitting so that this full connectivity is wasteful. The spatial extent of this local connectivity is a hyperparameter called the receptive field of the neuron (equivalently this is the filter size). By this way, what the convolution layer do comes out. Without fully connectivity, a small correlation window is being used to do a correlation calculation in a local region of the input. In LeNet-5, this kind of filter is 5*5 without zero padding and stride applied is 1. The explanation for convolution without zero-padding and stride are shown in **Figure** 2 and **Figure** 3.

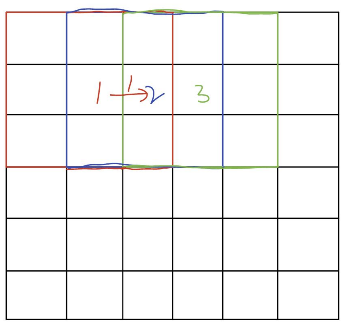


(a) Without zero-padding

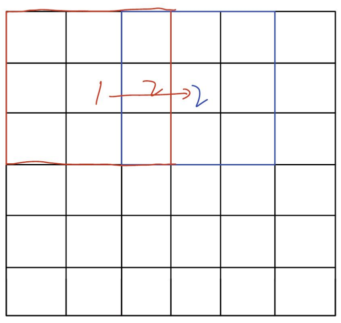


(b) With zero-padding

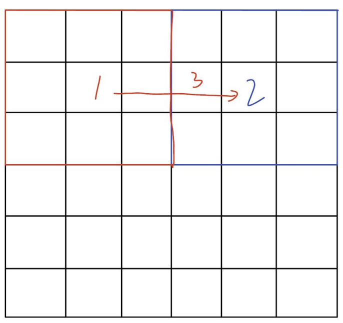
Figure 2: Comparison for using of zeropadding



(a) Stride = 1



(b) Stride = 2



(c) Stride = 3

Figure 3: Convolution with different stride

Having determined the size of convolution filter and the method to do convolution, how many filters we choose? This is the other hyperparameter, the depth of the output volume which is corresponding to the number of filters we would use. In LeNet-5, there are 6 filters with size 5*5*3 to use in first convolutional layer and 16 filters with size 5*5*6 in the second convolutional layer.
By this way, all hyperparameters are handled. Then, the training process is going on to determine the weight of each filter by backpropagation which will be introduced in details in next section. To be mentioned, parameter sharing scheme is used in Convolutional Layers to reduce the number of weights. If one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2). It allows the neurons in one output layer to share the same set ot parameter over the entire input.

`Max pooling Layer`

Between two successive Convolutional layer, there is a Pooling layer inserted to subsample the output of previous layer. The function is to progressively reduce the spatial size of the representation and the amount of parameters in the network in order to control overfitting. What the Pooling layer do is shown in **Figure** 4 .

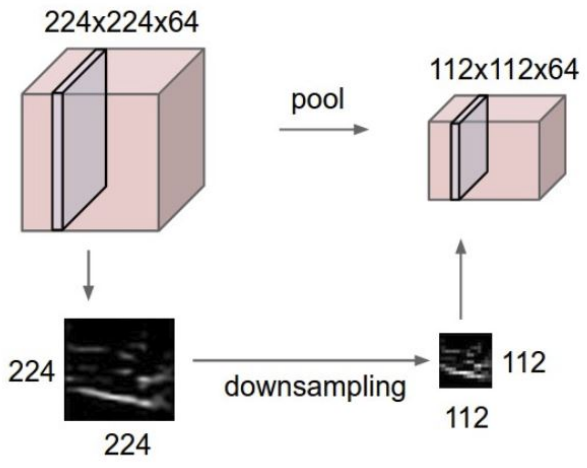


Figure 4: Pooling Layer

In LeNet-5, a 2*2 MAX window with stride = 2 is used to reduce the number of weights for the following layer. The MAX operation is that to keep the maximum in a 2*2 window and discard to other three value, which is shown in **Figure** 5.

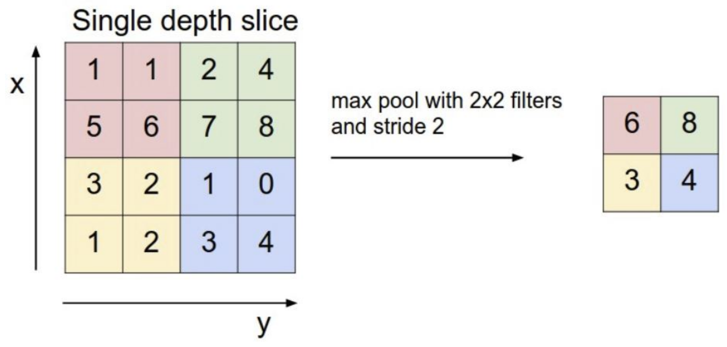


Figure 5: MAX operation

`Fully-connected Layer`

The Fully-connected layer is just like the normal hidden layer in regular ANN, working for classification. Neurons in a fully connected layer have full connections to all activations in the previous layer. To be mentioned, the fully connectivity make each neuron own all information about the input image. `Output Layer`
The last layer is output layer, which is always a high-dimension vector to represent the classification result. For example, $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ represents the first class.

`Activation function`

Having seen the main layer of CNN, we can also explicitly consider the activation function as a layer. The cell body and activation function are shown in **Figure** 6.

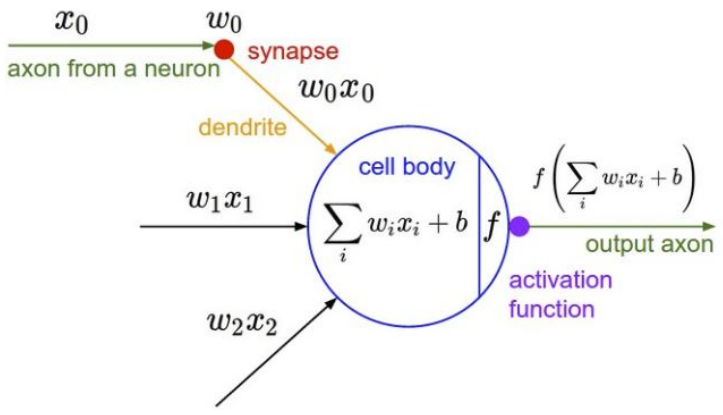


Figure 6: Cell body

In this figure, x1, x2 and x3 represent the value of neuron in the previous layers. w0, w1 and w3 represent the weights of the filter in this layer. And, the activations can hence be computed with a matrix multiplication followed by a bias offset. After that, we put this calculation result into non-linear activation function to get output axon. There are several different activation functions shown in **Figure** 7.

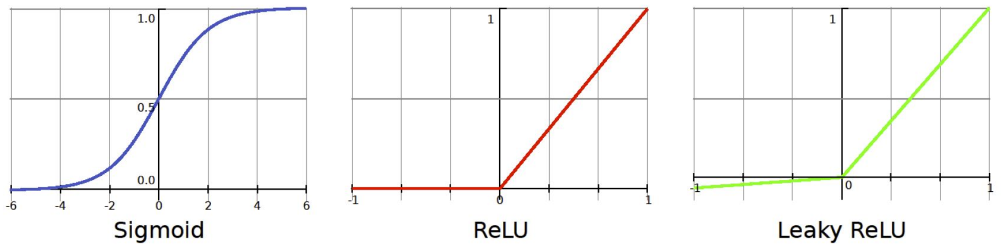


Figure 7: Activation function

In LeNet-5, RELU is used as elementwise activation function,

$$max(0, x)$$

By this way, non-linearity has been introduced accord with the real world. Compared to traditional neural network activation functions, such as logic functions (Logistic sigmoid) and tanh or other hyperbolic functions, linear rectification functions have the some advantages in accord with the brain in human beings.

`Softmax function`

Between the last fully-connected layer and output layer, Softmax function can by regarded as a explicitly layer. The formula is shown as followed,

$$s(x) = log(\frac{e^x}{\sum_{i=0}^{N} e^i})$$

Softmax provides the log probabilities of the output labels and it works for normalizing the outputs in (0, 1) to easily determine which class it belongs to.

1.2.2 Over-fitting issue

Overfitting means that the assumptions are too strict in training set. Avoiding overfitting is a core task in classifier design. For model training, several cases will cause over-fitting [2]:

- (1) The modeling sample is incorrectly selected, such as too few samples, incorrect sampling method or wrong sample labels, resulting in insufficient sample data to represent the predetermined classification rules.
- (2) The sample noise interference is too large so that the machine considers part of the noise as a feature and disturbs the preset classification rules.
- (3) There are too many parameters and the model complexity is too high.

Several techniques can be used to avoid over-fitting. For example, compared with traditional regular ANN whose all hidden layers are fully-connected layer causing large amount of parameter to train, CNN use Convolution layer and pooling layer first for feature extraction. Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting. Stochastic gradient descent can also avoid the overfitting in some degree. Also, from other reference, there are other methods to avoid over-fitting:

- (1) In the neural network model, the method of weight attenuation can be used, that is, each weight is reduced by a small factor in each iteration.
- (2) Select appropriate stopping training standards so that the training of the machine is at an appropriate level;
- (3) Obtain additional data for cross-validation;
- (4) Regularization, that is, when the objective function or the cost function is optimized, a regular term is added after the objective function or the cost function, and generally there are L1 regularity and L2 regularity.

Besides, dropout is very effective hyper-parameter to reduce overfitting, which is introduced in discussion section.

1.2.3 Why CNN work much better than other traditional methods?

Compared with tradition method requiring domain knowledge, human Adhoc trials and handcrafted features, CNN can find features systematically and automatically. It means feature extraction is just byproduct. What's more, parameter sharing in Convolutional layer make CNN handle high dimensional data.
Compared with traditional regular ANN, there are two main improvements for CNN:

- (1) Parameter sharing mechanism (parameters sharing)
  From the previous explanation, we know that filter is used to detect features. That feature is generally likely to appear in more than one place, such as "vertical boundary". It can be seen that the parameter sharing mechanism greatly reduces the number of parameters of our network. By this way, we can train better models with fewer parameters, typically with less effort, and it can effectively avoid overfitting. Similarly, because the parameters of filters are shared, even if the image is subjected to a

certain translation operation, we can still recognize the feature, which is called “translation invariance”. Therefore, the model is more stable.

- (2) The sparsity of connections (sparsity of connections)
  As we see from the convolution operation, any neuron in the output image is only related to a part of the input image. However, in the traditional neural network, since all are connected, any neuron of the output is affected by all input neuron. By this way, the recognition effect of the image is greatly reduced. In comparison, each region has its own unique characteristics, and we do not want it to be affected by other regions.

It is precisely because of these two advantages that CNN has surpassed the traditional ANN and opened up a new era of neural networks. Although the reason implicitly why CNN work so well is difficult to imply, from the aspect of revolution in neural network from ANN to CNN, big data and strong ability to compute make it perform well. Actually, neural network is not new word. Even for deep learning, so called three pioneers Bengio, Hinton and LeCun proposed it almost 20 years ago. It can be said that it is really a necessary trend for CNN work so well today with the great development of training data size and improvement of computation ability. However, it is really hard to say whether CNN and other related deep learning techniques will be dominate or back to history. In March 27th, 2019, 3 Pioneers in Artificial Intelligence win Turing Award. The main contribution is on deep learning or kind of CNN, which make CNN keep dominating the popular method for AI [3].

In image classification, CNNs can be thought of automatic feature extractors from the image. While if I use a algorithm with pixel vector I lose a lot of spatial interaction between pixels, a CNN effectively uses adjacent pixel information to effectively downsample the image first by convolution and then uses a prediction layer at the end. This concept was first presented by Yann le cun in 1998 for digit classification where he used a single convolution layer. It was later popularized by Alexnet in 2012 which used multiple convolution layers to achieve state of the art on imagenet. Thus making them an algorithm of choice for image classification challenges henceforth.

1.2.4 Loss function and classical backpropagation optimization

Before talking about Backpropagation, let’s look at loss function. In mathematical optimization, statistics, econometrics, decision theory, machine learning and computational neuroscience, a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function [4].

In CNN, if the output of a training sample is y', we can define the cost function as the Euclidean distance between y' and its corresponding unit vector. For example, if we know the ground-truth of the input image should be $y = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, which means this input image should be assigned to class A. However, the normalized result by log soft max function is $y' = [0.65, 0, 0, 0.1, 0, 0, 0.2, 0, 0, 0.05]$. Then we can define cost function to be,

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{10}(y_i - y_i')^2$$

Generally, we use mean square error to be loss function or cost function. In CNN, we use w_i to represents weights and b_i to represents bias parameter. Then all parameters should be,

$$\text{Network parameters} \quad \theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$$

By this way, having defined the cost function, we start to train. First, it is about initialization and random initialization is often used to get the starting parameter θ^0. With the following procedures, we iterate the value of θ again and again until it converges.

$$\begin{tikzcd}\begin{array}{c}\text{Starting}\\\text{Parameters}\end{array} \quad \theta^0 \arrow[r] & \theta^1 \arrow[r] & \theta^2 \arrow[r] & \cdots\cdots\end{tikzcd}$$

The method used here is gradient descent and the formula is shown as followed,

$$\theta^1 = \theta^0 - \eta\nabla\mathrm{L}(\theta^0)$$

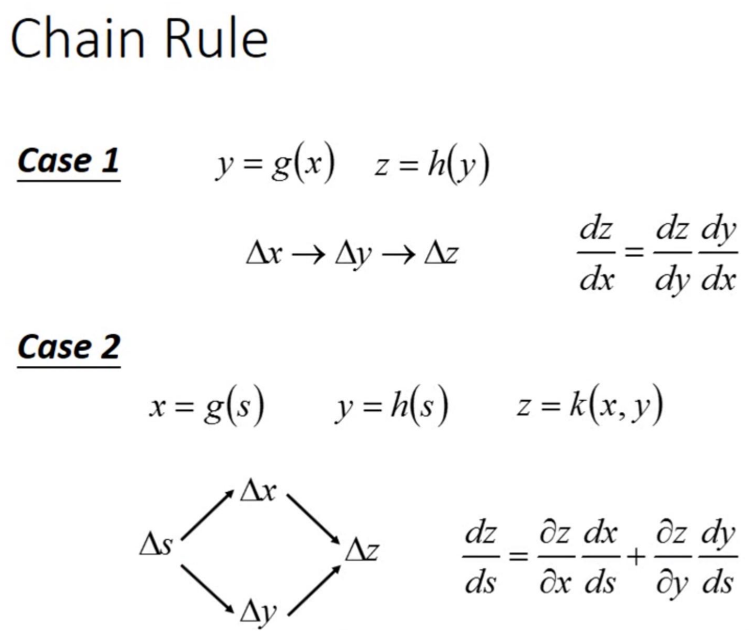
where η is learning rate, one of the important hyperparameter for CNN(High learn rate will cause fast converge but overfitting) and $\triangledown L(\theta)$ is partial derivation for each parameter,

$$\nabla\mathrm{L}(\theta) = \begin{bmatrix} \partial\mathrm{L}(\theta)/\partial w_1 \\ \partial\mathrm{L}(\theta)/\partial w_2 \\ \vdots \\ \partial\mathrm{L}(\theta)/\partial b_1 \\ \partial\mathrm{L}(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

Sometimes, we use stochastic gradient descent to replace gradient descent in order to avoid overfitting. The formula is shown as followed,

$$\theta=\theta-\alpha\ \nabla\theta\ J(\theta;x(i),y(i))$$

Generally, there may be a large amount of parameter to train. So, the vector of partial deviatoin will be high dimension. How to get the result of partial deviatoin for each parameter is where backpropogation been used. Since CNN is consisted a sequence of several layers, no matter linear convolution operation in convolution layer or non-linear activation function, if we want to get the partial derivation for any parameter, chain rule is used as the chain to "pass" the intermediate derivation result to the final one. The diagram for chain rule is shown in **Figure** 8.



Chain Rule

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

$$x = g(s) \qquad y = h(s) \qquad z = k(x, y)$$

$$\begin{tikzcd} & \Delta x \arrow[dr] & \\ \Delta s \arrow[ur] \arrow[dr] & & \Delta z \\ & \Delta y \arrow[ur] & \end{tikzcd} \qquad \frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

Figure 8: Chain Rule

Having introduced all basic knowledge for CNN training, let's look at what backpropogation is and why it is called "back".

First, let's look at the small demo network shown in **Figure** 9.

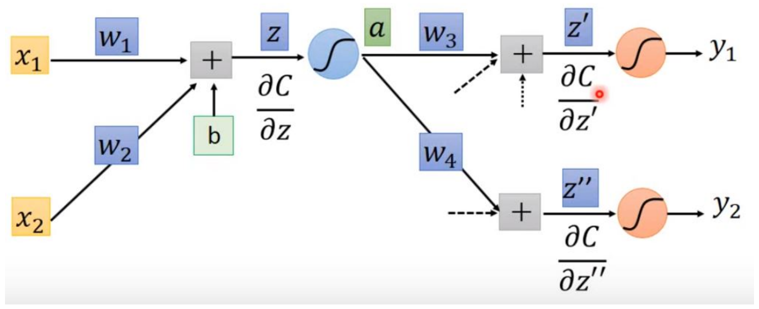

x_1
w_1
z
a
w_3
z'
y_1
$\frac{\partial C}{\partial z}$
$\frac{\partial C}{\partial z'}$
w_2
b
w_4
z''
y_2
x_2
$\frac{\partial C}{\partial z''}$

Figure 9: Demo network

Assume we want to calculate the partial derivation of w_1, we need to calculate partial derivation of z. The formula is shown as follow,

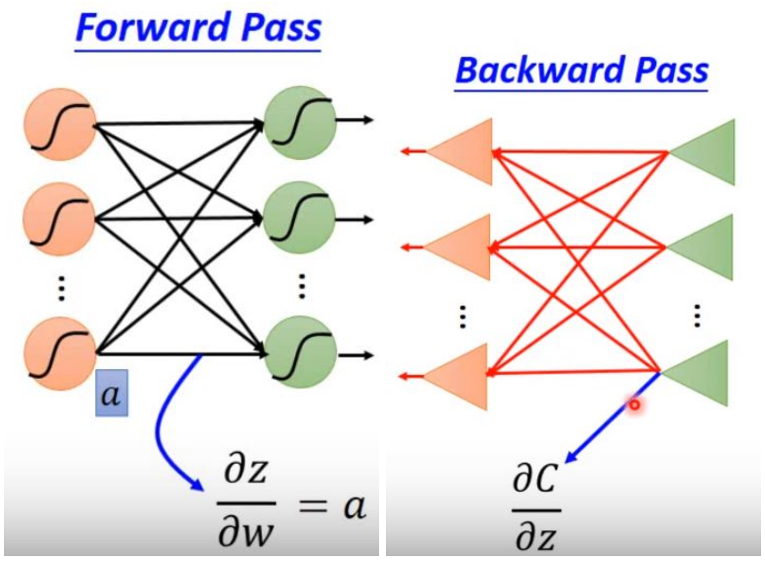
$$\frac{\partial C}{\partial w} =? \quad \frac{\partial z}{\partial w}\frac{\partial C}{\partial z}$$

(Chain rule)

Also, by chain rule, if we want to calculate partial derivation of z, we need to know partial derivation of z' and z''. The formula is shown as follow,

$$\frac{\partial C}{\partial z} = \sigma'(z)\left[w_3 \frac{\partial C}{\partial z'} + w_4 \frac{\partial C}{\partial z''}\right]$$

By this way, if we want to calculate the partial derivation of the parameter in first layer followed by input layer, we need know the other partial derivation of parameters in following layers. We know the size of parameter is huge so that the order to calculate forward is really wasteful for computation. Thus, why we calculate partial derivation backward? By this way, backpropagation comes out, which is shown in **Figure** 10.


Forward Pass
a
$\frac{\partial z}{\partial w} = a$
Backward Pass
$\frac{\partial C}{\partial z}$

(a) Forward pass (b) Backward pass

Figure 10: Backpropagation

Backpropagation means that we calculate the partial derivation of the parameter in last layer first and then propagate result from back to front. In this order, repeated computation will be avoided. Thus, we can regard backpropagation as a strategy to optimize the calculation of those partial derivation.
If we don't use backpropogation, this is very redundant because many paths are repeatedly accessed. For a neural network in a depth model with tens of thousands of weights, the amount of computation caused by such redundancy is quite large. Similarly, using the chain rule, the BP algorithm wisely avoids this redundancy, and it can only find the partial derivative value of the vertex to all the lower nodes for each path. As the name of the Back Propagation (BP) algorithm says, the BP algorithm is reverse (top to bottom) to find the path.

2 Train LeNet-5 on MNIST Dataset

2.1 Architecture of LeNet-5

Having introduced the basic components of CNN, let's look at the architecture of LeNet-5. The weights training in order by procedures is shown in **Table 1**. All in all, there are 60806 parameters for training [5].

Table 1: Statistics for training parameter in LeNet-5

Layer	Parameters	Neurons
Conv1	156	28*28*6=4704
Pooling1	0	14*14*6=1176
Conv2	1516	10*10*16=1600
Pooling2	0	5*5*16=400
Fully connected1	48120	120
Fully connected2	10164	84
Fully connected3	850	10

`Input layer`

Since the size of input images in MNIST is 32*32*3, the input layer consists of 32*32 neurons.

`Convolution layer1`

In this layer, we set the receptive field as 5*5 with no zero padding and stride = 1. And, the size of filter bank is 6 so that an image of size 32*32 is transformed into 28*28 with depth 6. The height and width just minus 4 (32-4=28) because of no padding using. The number of neurons is 28*28*6. With the using of parameter sharing, the number of weights in this layer is 156, $(5 * 5 + 1) * 6$. 1 represent the bias parameter and there are 6 filters in filter bank.

`Max-pooling layer1`

By downsampling with MAX function in 2*2 window of stride = 2, the height and width will be divided by 2 to be 14 so that the size after this layer will be 14*14*6. There are no weights.

`RELU Activation1`

In this layer, non-linearity is introduced by thresholding at zero. There is no changes for input size and no weight to learn.

`Convolution layer2`

Compared with the first convolution layer, the input size is 14*14*6 this time and the output layer is 10*10*16 so that there are 16 filters in filter bank. They are 6 filters with size 5*5*3, 1 filters with size 5*5*6 and two classes of filters with size 5*5*4. One class has 3 filters and the other has 6 filters. Thus, the total number of filters is 16. Having done convolution, the output size is 10*10*16. The size of training weights is 1516, $6 * (3 * 5 * 5 + 1) + 6 * (4 * 5 * 5 + 1) + 3 * (4 * 5 * 5 + 1) + 1 * (6 * 5 * 5 + 1)$.

`Max-pooling layer2`

The size of window using MAX function is still 2*2 so that the size after this layer will be 5*5*16 and there are no weights.

`RELU Activation2`

Similar to the first RELU Activation function, thresholding to zeros is used. Upon here, the total number of neurons has decreased from 1024(32*32) to 400(5*5*16). Except for this dimension reduction, feature extraction has been finished.

`Fully-connected layer1`

The input size is 5*5*16 and the maximum receptive field is used in fully connected layer. In this layer, we use 120 filter with size 5*5*16 to get 120 response so that there are 120*(16*5*5+1) = 48120 parameters to train. Also, RELU Activation is used in their output to model non-linearity. At this time, for these three fully-connected layer, I will not introduce the RELU activation as a separable layer anymore.

`Fully-connected layer2`

The input size is 120 and the maximum receptive field is used in fully connected layer. In this layer, we use 84 filter with size 120 to get 84 response so that there are 84*(120+1) = 10164 parameters to train. Also, RELU activation is used in outputs.

`Fully-connected layer3`

The input size is 84 and the maximum receptive field is used in fully connected layer. In this layer, we use 10 filter with size 84 to get 10 response so that there are 10*(84+1) = 850 parameters to train. Also, RELU activation is used in outputs.

`Output layer/Soft max function`

Between the last fully-connected layer and output layer, Softmax function can by regarded as a explicitly layer. Softmax provides the log probabilities of the output labels and it works for normalizing the outputs in (0, 1) to easily determine which class it belongs to.

2.2 Approach and procedures

The general flow diagram for the training and testing process is shown in **Figure** 11.

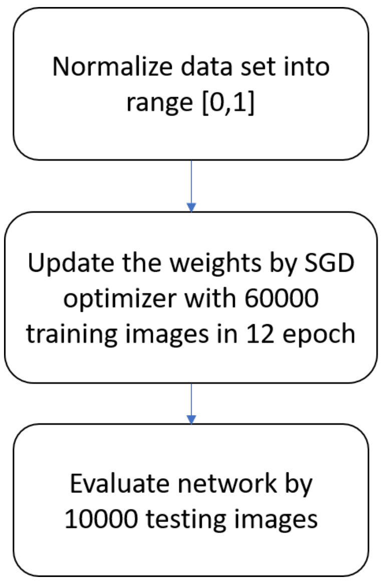

Normalize data set into range [0,1]
Update the weights by SGD optimizer with 60000 training images in 12 epoch
Evaluate network by 10000 testing images

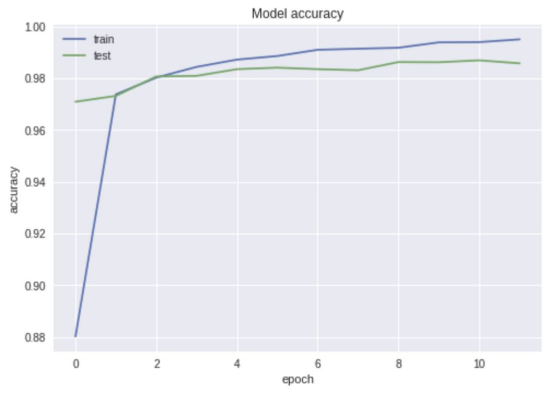
Figure 11: Flow diagram

2.3 Experimental Results

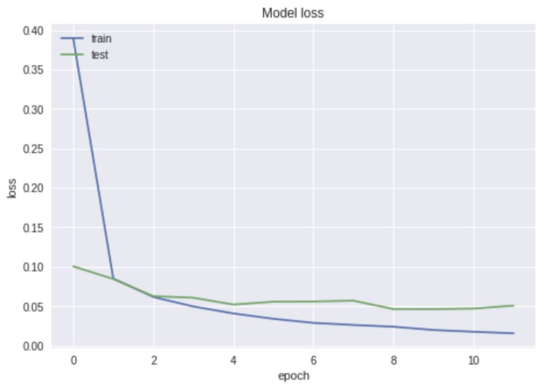
To be mentioned first, in LeNet-5 there are several hyperparameters which can influence the result of CNN. They are epoch, batch size and dropout. Also, the choice of optimizer will significantly affect the convergence result. The hyperparameters we can set are learning rate, decay and momentum. More details about different parameter will be explained in next section. In this section, I will give several result in different representative parameter settings.

I use the notation (epoch,batch size,dropout,learning rate,decay,momentum) to simplify the following figure representation. And, the following figures about accuracy and loss curve are in the validation split of 0.2, which means if the total number of training set is 60000, 80 percent of this set 48000 images are used for training and 20 percent of this set 12000 images are used for evaluation in each epoch.

The **Figure** 12 is shown in default settings. And then, we compare other figures with this figure by changing specific hyperparameter.



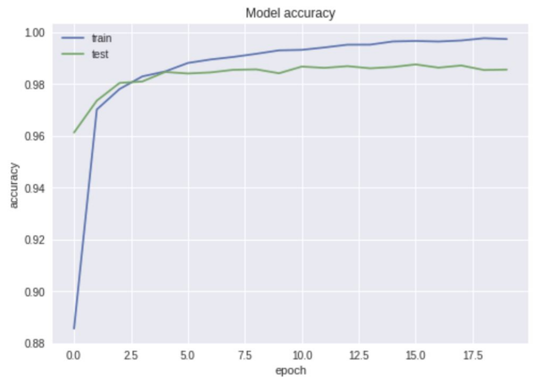
(a) Accuracy curve



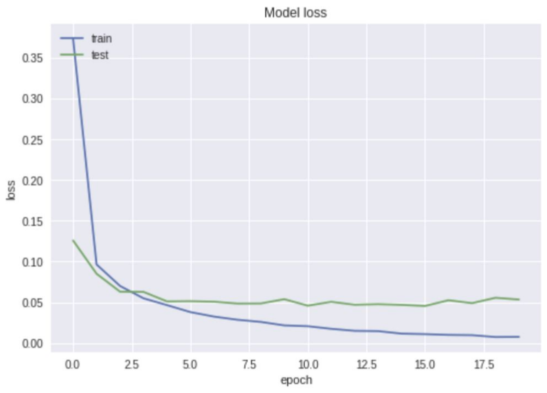
(b) Loss curve

Figure 12: (12,128,N/A,0.01,1e-6,0.9) Test accuracy = 0.9882 Training accuracy = 0.9945

The epoch is changed from 12 to 20 in **Figure** 13 .



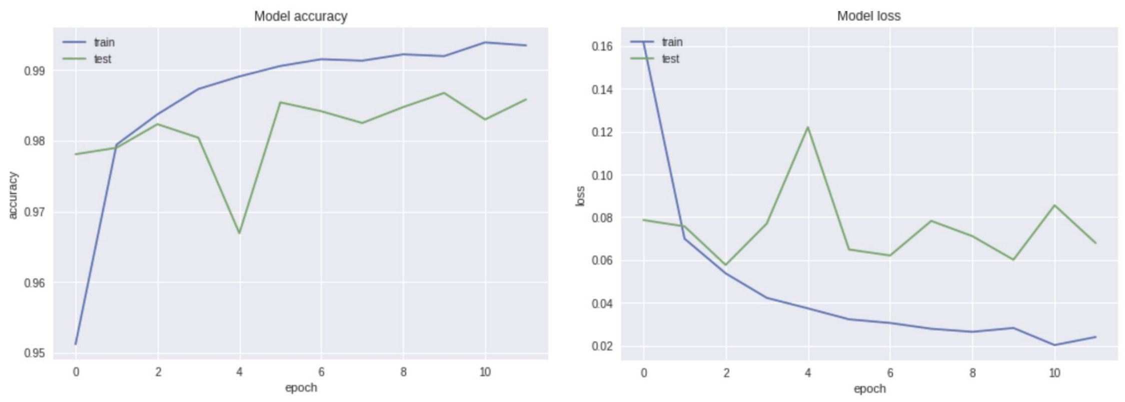
(a) Accuracy curve



(b) Loss curve

Figure 13: (20,128,N/A,0.01,1e-6,0.9) Test accuracy = 0.9868 Training accuracy = 0.9956

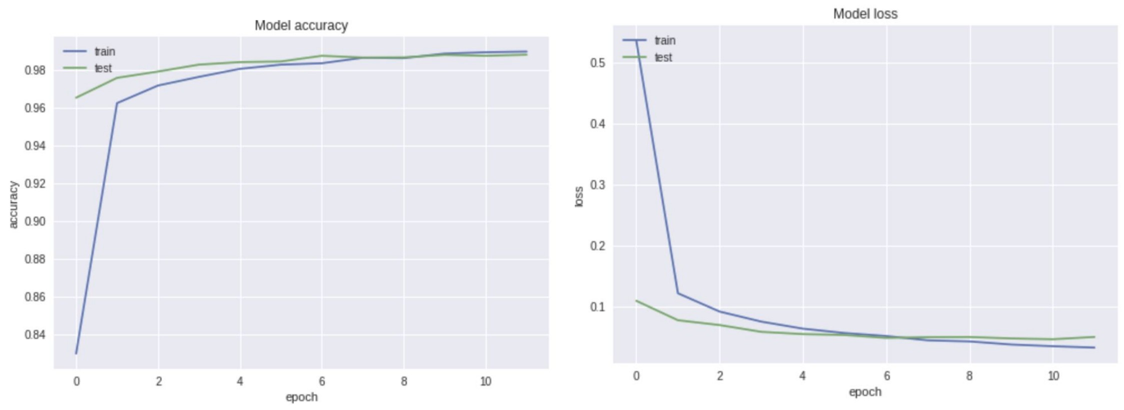
The batch size is changed from 128 to 10 in **Figure** 14.



(a) Accuracy curve (b) Loss curve

Figure 14: (12,10,N/A,0.01,1e-6,0.9) Test accuracy = 0.987 Training accuracy = 0.996

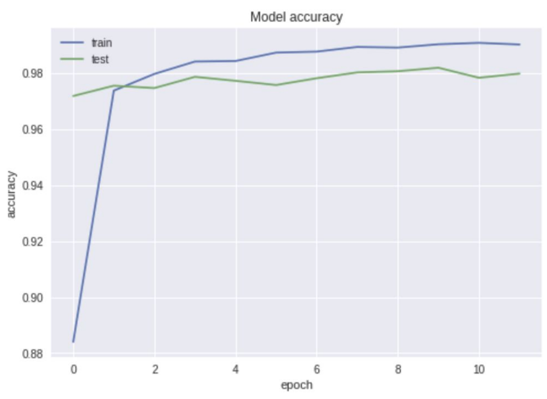
The dropout is changed from "not available" to 0.2 in **Figure** 15. And, I just drop 20 percent neurons in first fully connected layer.

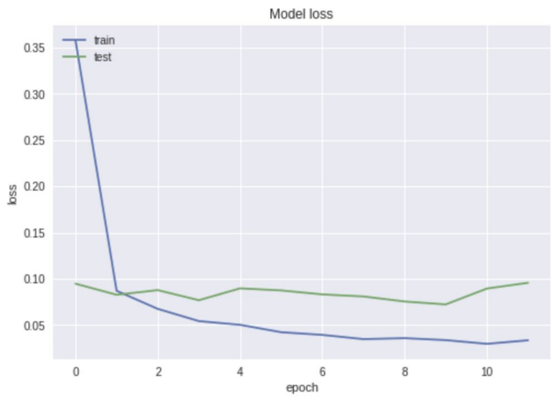


(a) Accuracy curve (b) Loss curve

Figure 15: (12,128,0.2,0.01,1e-6,0.9) Test accuracy = 0.9885 Training accuracy = 0.9925

The learning rate is changed from 0.01 to 0.1 in **Figure** 30.

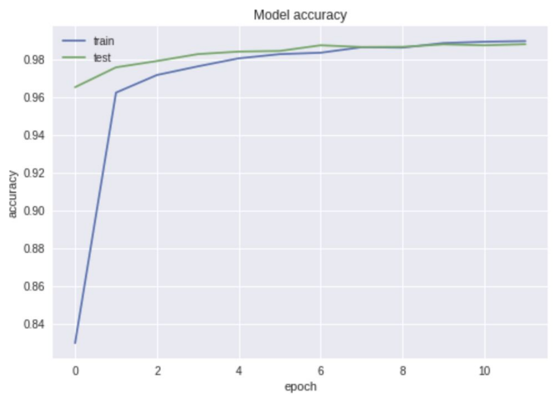


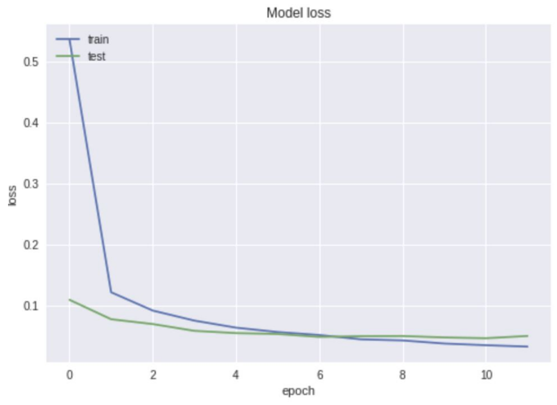


(a) Accuracy curve (b) Loss curve

Figure 16: (12,128,N/A,0.1,1e-6,0.9) Test accuracy = 0.9805 Training accuracy = 0.98815

In summary, for the five accuracy results in different setting, the mean of accuracy for testing data set is 0.9862 and variance is 8.556E-6. The mean of accuracy for training data set is 0.99335 and variance is 8.234E-6. Finally, the best result is obtained in setting (12,128,0.2,0.01,1e-6,0.9). The best result is shown in **Figure** 17.





(a) Accuracy curve (b) Loss curve

Figure 17: (12,128,0.2,0.01,1e-6,0.9) Test accuracy = 0.9885 Training accuracy = 0.9925

To make the result more plausible, I make another five trials to test the accuracy in this best parameter setting, which is shown in **Table** 2. What's more, the mean is 0.9884 and variance is 1.46e-06 for this best parameter testing. By this way ,the result is reliable and credible.

Table 2: Accuracy for different five trials in best parameter setting

No.	Accuracy (Original images)
1	0.9872
2	0.9901
3	0.9935
4	0.9871
5	0.9881

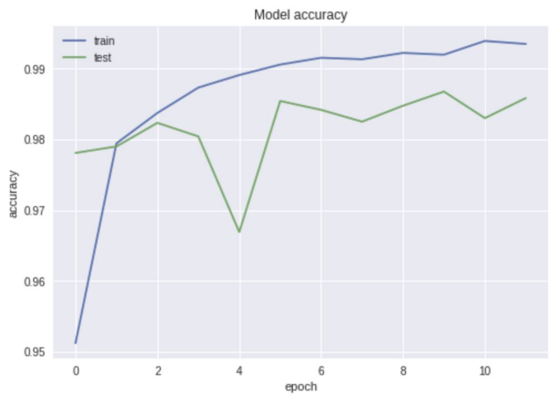
2.4 Discussion

2.4.1 Epoch

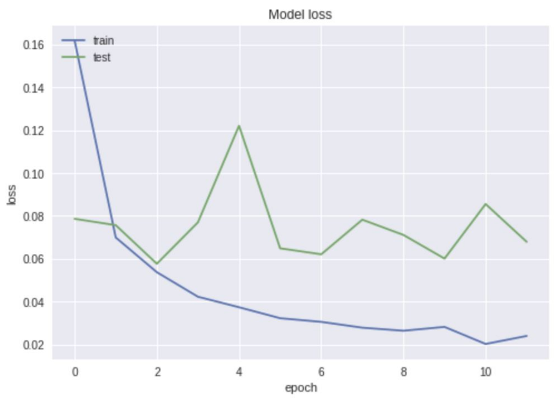
One epoch means one forward pass and one backward pass of all training examples. To be mentioned, there is no setting about iteration which represents number of passes, each pass using [batch size] number of examples. As the epoch process, the accuracy curve tend to converge into one specific value and this value is the final accuracy. Thus, the small number of epochs is not enough for the curve to converge, but too large number of epoch may be wasteful if the improvement of accuracy is very limited in backward epoch. An appropriate number of epoch is very important number for the model to get the reasonable accuracy curve without large time expense.

2.4.2 Batch size

The size of batch is involving with the mini-batches gradient descent, which means within the batch size of samples we do one time backprogogation to update the weights. If the batch size is small, the time for one epoch will be increasing and some vibrations will occur in accuracy curve shown in **Figure** 18. What's more, too small size of batch is difficult for model to converge.



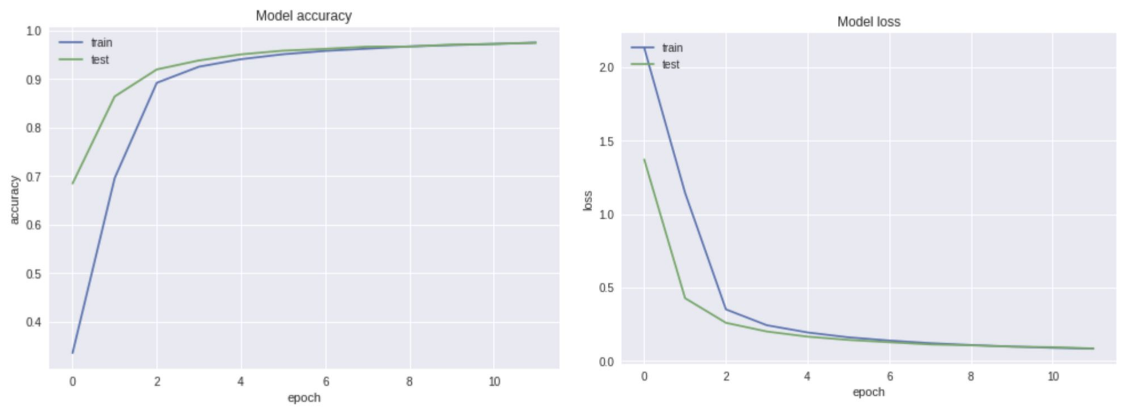
(a) Accuracy curve



(b) Loss curve

Figure 18: Batch size is 10(12,10,N/A,0.01,1e-6,0.9) Test accuracy = 0.987 Training accuracy = 0.996

However, too huge size of batch is also not good. Memory utilization has increased, but memory capacity may not hold up. When batch Size is increasing to a certain extent, its determined downward direction will not be changed substantially. Just like **Figure** 19, in the last several epochs, the accuarcy cannot be more accurate anymore because at this time gradient descent don't make sense.



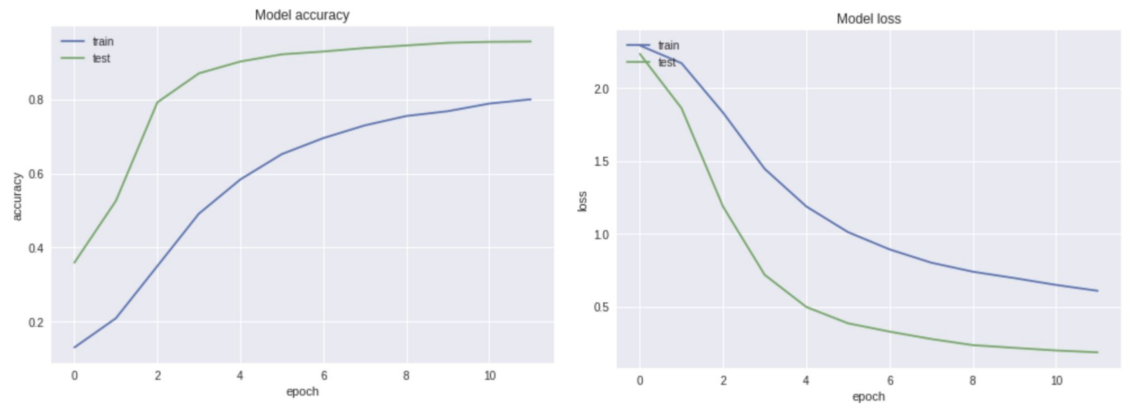
(a) Accuracy curve (b) Loss curve

Figure 19: Batch size is 10(12,1280,N/A,0.01,1e-6,0.9) Test accuracy = 0.987 Training accuracy = 0.996

Batch size is the number of training examples in one forward and backward pass. The higher the batch size, the more memory space we'll need. If the data set is small, it can be training in full Batch. There are at least two advantages. First, the direction determined by the full data set is better to represent the sample population. Second, because the gradient values of different weights are very different, it is difficult to select a global learning rate. Full Batch Learning can use Rprop to update each weight individually based on gradient symbols only.
For larger data sets, the above two benefits have became two disadvantages. First, with the massive growth of data sets and memory limitations, it is becoming less and less feasible to load all the data at once. Second, due to the sampling differences between the batches, the gradient correction values cancel each other and cannot be corrected, which is followed by the compromise with RMSProp.

2.4.3 Dropout

Dropout is used to avoid overfitting by ignore some neurons in fully-connected layer. However, if we use two much dropout which means we drop too much parameter, underfitting will happen, just like in **Figure** 20.

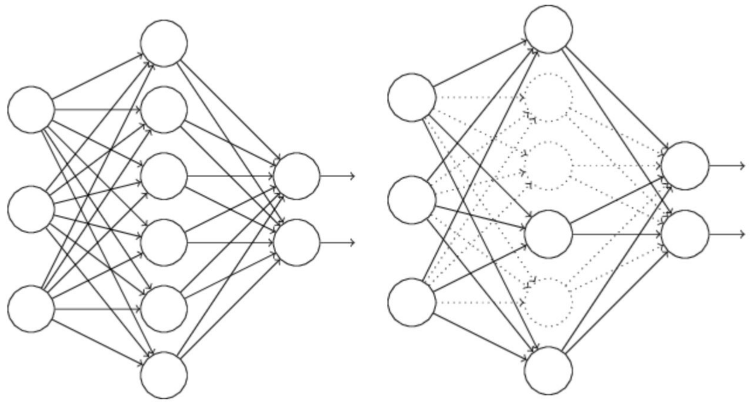


(a) Accuracy curve

(b) Loss curve

Figure 20: Dropout is 0.8 (12,128,0.8,0.01,1e-6,0.9) Test accuracy = 0.987 Training accuracy = 0.996

Why does dropout help prevent overfitting? It can be simply explained that the training process using dropout is equivalent to training a large number of neural networks with only half of the hidden layer units, which is shown in **Figure** 21, and each such half network can give a classification. As a result, some of these results are correct and some are wrong. As the training progresses, most of the half networks can give correct classification results, so a small number of mis-classification results will not have a big impact on the final result.



(a) Original netword

(b) Network with dropout

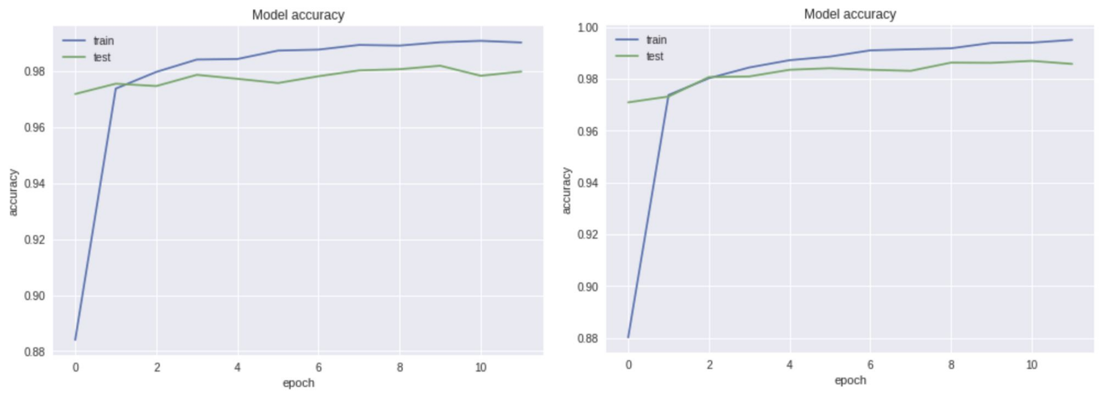
Figure 21: Dropout

What's more, this "comprehensive averaging" strategy is usually effective in preventing overfitting problems. The entire dropout process is equivalent to averaging many different neural networks. Different networks produce different over-fittings, and some “reverse” fittings cancel each other out to reduce the over-fitting as a whole.

Also, dropout forces two neurons not necessarily to appear in a dropout network each time. Such updates of weights no longer rely on the interaction of implicit nodes with fixed relationships. The network is forced to learn more robust features that also exist in random subsets of other neurons. In other words, if our neural network is making some kind of prediction, it should not be too sensitive to some specific clues. Even if it loses certain clues, it should be able to learn some common features from many other clues.

2.4.4 Learning rate

The learning rate determines the speed at which the weight is updated. If the setting is too large, the result will exceed the optimal value. Too small learning rate will make the falling speed too slow. The comparison with different learning rate is shown in **Figure** 22.



(a) Learning rate = 0.1 (b) Learning rate = 0.01

Figure 22: Learning rate

From the figure, we can see that bigger learning rate create much many fluctuation. But sometimes smaller learning rate may cause over-fitting and slow speed to converge.
Adjusting parameters only by human intervention requires constant modification of the learning rate. It's good to set learning rate adaptive by epochs. By this way, the automotive learning rate adjustment is done by learning rate decay and momentum.

2.4.5 Learning rate Decay

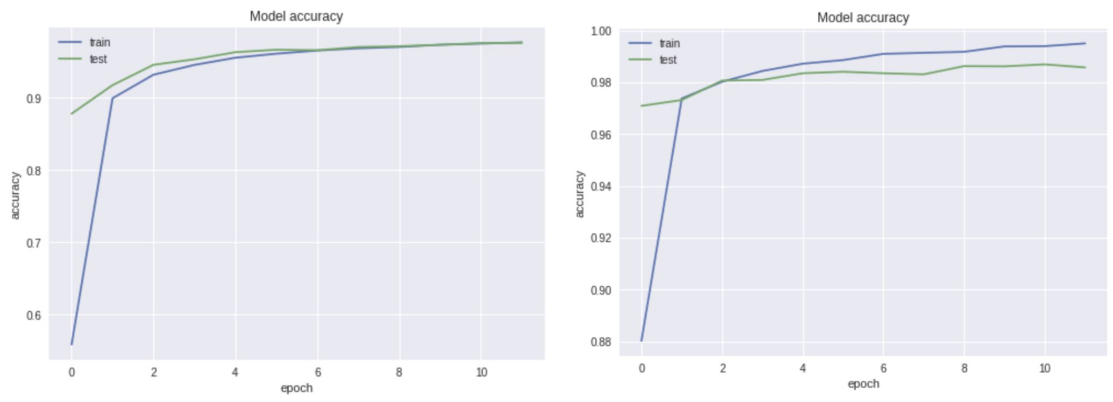
During the training process, the dynamically changing learning rate is generally set according to the number of training rounds. In SGD, the formula is shown as followed,

$$LearningRate = LearningRate * 1/(1 + decay * epoch)$$

At the beginning of training, the learning rate is preferably 0.01 - 0.001. After a certain number of rounds, learning rate gradually slows down. Near the end of training, the attenuation of the learning rate should be more than 100 times. Learning rate decay over each update. This method is to improve the SGD optimization ability, specifically to reduce the learning rate at each iteration.

2.4.6 Momentum

Momentum is a parameter that accelerates SGD in the relevant direction and dampens oscillations. Momentum is derived from Newton's law. The basic idea is to find the optimal "inertia" effect. When there is a flat region in the error surface, SGD can learn faster. The comparison about different momentums is shown in **Figure** 23.



(a) Momentums = 0.1 (b) Momentums = 0.9

Figure 23: Momentums

From the figure, we can see that the proper momentum is really helpful to increase the effects of learning rate. We know the disadvantage of SGD is that its update direction is completely dependent on the current batch. So its update is very unstable. Momentum is used to solve this problem, which simulates the inertia of an object when it is moving, that is, it retains the direction of the previous update to a certain extent when updating, and uses the gradient of the current batch to fine tune the final update direction. In this way, stability can be increased to a certain extent, so that learning is faster, and there is also the ability to get rid of local optimum.

In machine learning or pattern recognition, overfitting occurs, and when the network gradually overfitting, the network weight gradually becomes larger. Therefore, in order to avoid overfitting, a penalty term is added to the error function. The commonly used penalty is the square of the weight of ownership. Multiply by the sum of a decay constant. It is used to punish large weights.

2.4.7 Summary of CNN

Some ML practitioners who have had previous exposure to Neural Networks (ANN), would have the first impression that Deep Learning is nothing more than ANN with multiple layers. Furthermore, the success of DL is more due to the availability of more data and the availability of more powerful computational engines like Graphic Processing Units (GPU).

Convolutional networks have the following advantages over image processing in general neural networks. a) Input image and network topology can kiss very well. b) Feature extraction and pattern classification are performed simultaneously. c) Weight sharing can reduce the training parameters of the network.

A convolutional network is essentially an input-to-output mapping that learns a large number of mappings between input and output without the need for any precise mathematical expression between input and output.

2.4.8 Limitations of LeNet-5

CNN is able to derive an effective representation of the original image, which enables CNN to recognize the laws above the vision directly from the original pixels with minimal pre-processing. However, due to the lack of large-scale training data at the time, the computing power of the computer could not keep up. LeNet-5's processing of complex problems was not satisfactory.

Since 2006, many methods have been designed to overcome the difficulty of training deep CNN. Among them, the most famous is Krizhevsky et al. proposed a classic CNN structure and made a major breakthrough in image recognition tasks. The overall framework for its approach is called AlexNet, similar to LeNet-5, but to be deeper.

3 Apply trained network to negative images

3.1 Abstract and Motivation

As we see, the above-mentioned network work pretty well in classify the MNIST data set. But we still have a straight forward question: does the LeNet-5 really understands the handwritten digits as well as human beings? There is no difficulty for even a new-born. As he can recognize the original images in **Figure** 24, he can definitely recognize the negative images shown in **Figure** 25. What about CNN LeNet-5?

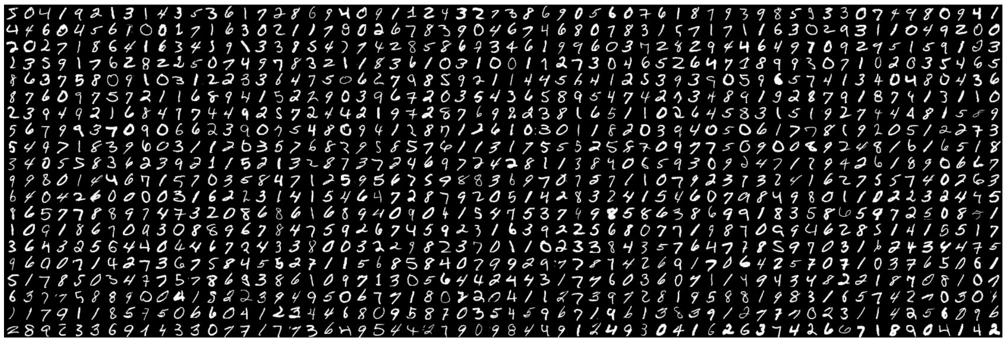


Figure 24: Origin images

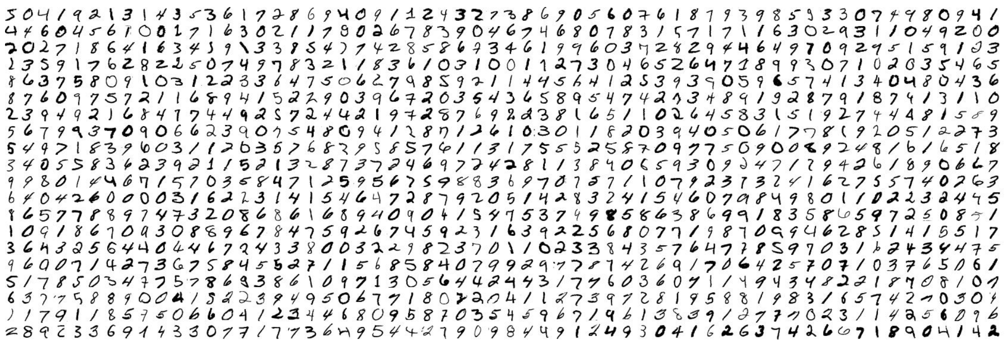
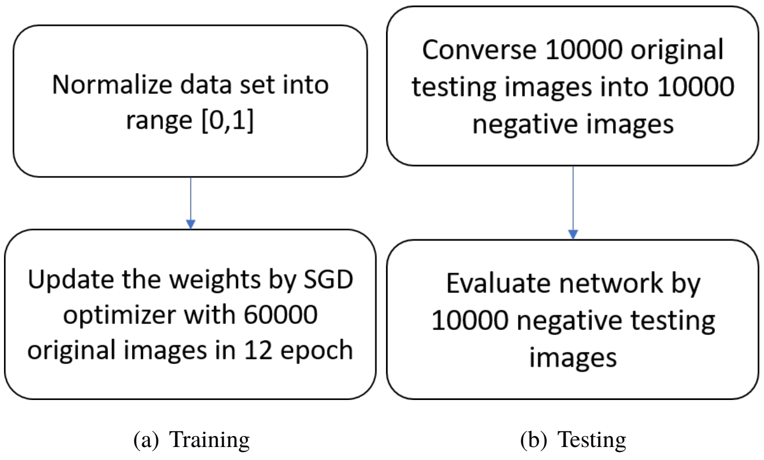


Figure 25: Negative images

3.2 Approach and procedure

First, let's test the accuracy in the negative test images using the LeNet-5 trained before. The procedures are shown in **Figure** 26.

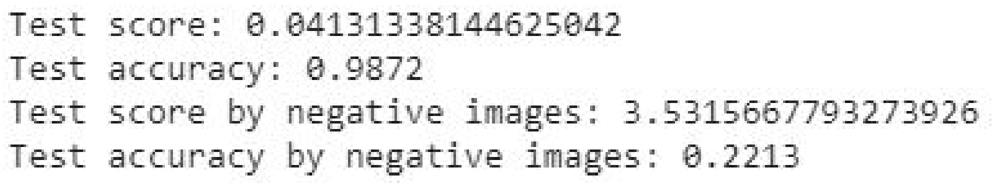


Normalize data set into range [0,1]
Update the weights by SGD optimizer with 60000 original images in 12 epoch
Converse 10000 original testing images into 10000 negative images
Evaluate network by 10000 negative testing images
(a) Training
(b) Testing

Figure 26: Flow diagram

3.3 Experimental Results

Right there, I give one evaluation result in **Figure** 31.



```
Test score: 0.04131338144625042
Test accuracy: 0.9872
Test score by negative images: 3.5315667793273926
Test accuracy by negative images: 0.2213
```

Figure 27: Negative images test

What's more, five sets of accuracy are obtained in different parameter, which is shown in **Table** 3.

Table 3: Statistics for testing by negative images

No.	Accuracy (Original images)	Accuracy (Negative images)
1	0.9872	0.2213
2	0.9901	0.3919
3	0.9895	0.2642
4	0.9871	0.2571
5	0.9881	0.2664

After calculation, we get that mean is 0.2801 and variation is 0.003385.

3.4 Discussion

The network obtained by whole bunch of original images is heavily dependent on the pixel values rather than the structure of different number. For human beings, we recognize these numbers mainly by inn er structure differences. Thus, for a new-born, even he haven't seen any negative images and his parent just teach him with original images, I think he can recognize the negative images pretty well.
From the above-mentioned result, we can conclude that network hasn't learned anything about structure of numbers. It just makes classification mainly by actual pixel value of images. For example, for the original images(black background and white number), after training, the network will regard the black pixels as kind of "don't care" neurons and neglect them when classification. It just focus on white pixel which give the structure of number. However, when the negative images(white background and black number) come in, the trained network by whole bunch of original images cannot any clues from white pixels so that the classification is not good.

4 New network to recognize both original and negative images

4.1 Approach and procedure

One straightforward idea to let the network recognize both original and negative images from the MNIST test dataset is using both original and negative images as training set. By this way, maybe the network can learning something new to classify them. The flow diagram is shown in **Figure** 28.

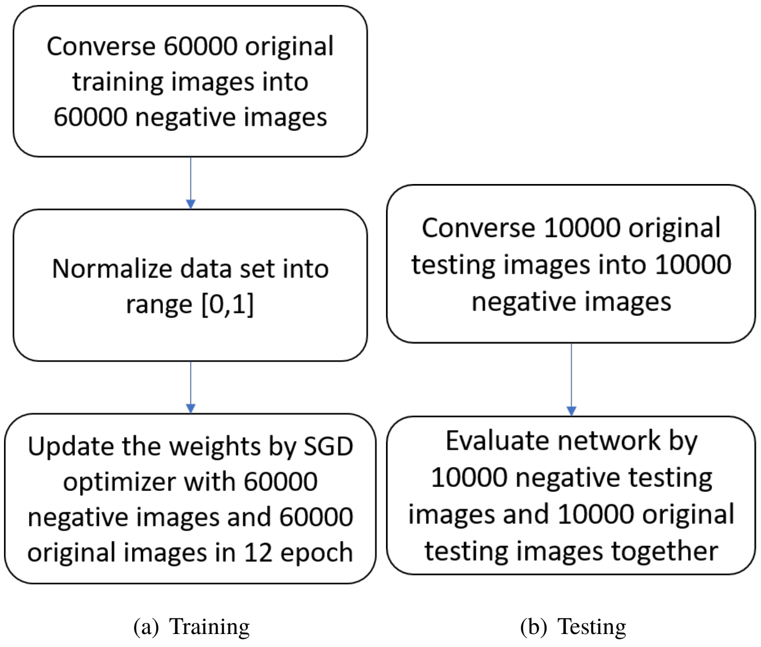
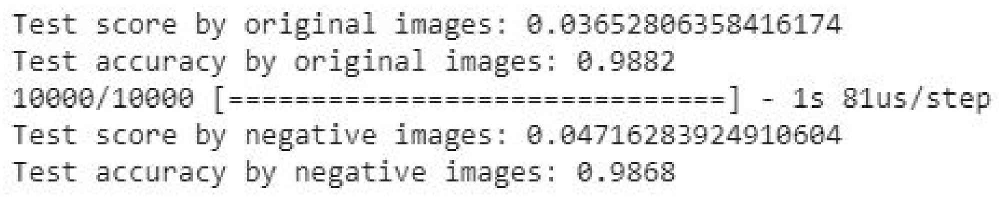

Converse 60000 original training images into 60000 negative images
Normalize data set into range [0,1]
Update the weights by SGD optimizer with 60000 negative images and 60000 original images in 12 epoch
Converse 10000 original testing images into 10000 negative images
Evaluate network by 10000 negative testing images and 10000 original testing images together
(a) Training
(b) Testing

Figure 28: Flow diagram

4.2 Experimental Results

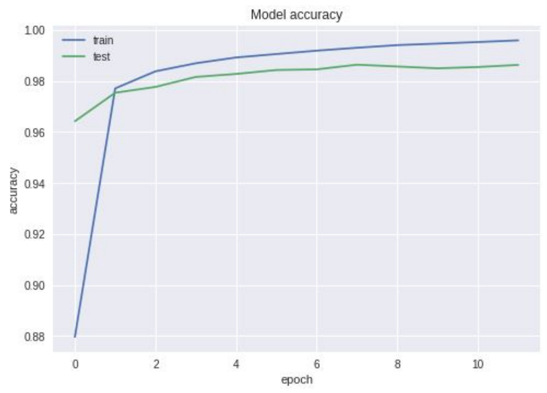
After training the new network, I first evaluate it by original images and negative images separately. The results are shown in **Figure** 31.



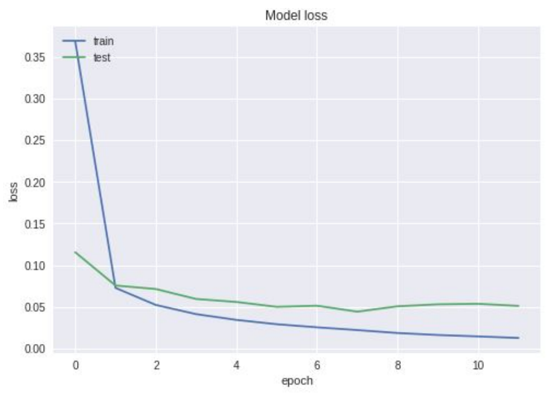
```
Test score by original images: 0.03652806358416174
Test accuracy by original images: 0.9882
10000/10000 [==============================] - 1s 81us/step
Test score by negative images: 0.04716283924910604
Test accuracy by negative images: 0.9868
```

Figure 29: Separable test

And then, the test by data set which is the combination of 10000 original images and 10000 negative images is shown in **Figure** 30 and **Figure** 31.

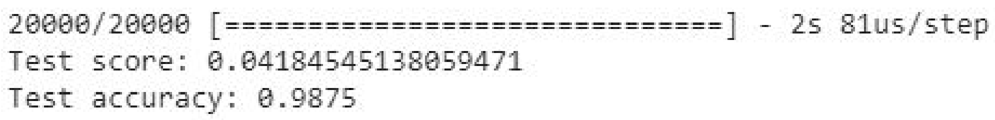


(a) Accuracy curve



(b) Loss curve

Figure 30: Testing result for network trained by 120000 images



```
20000/20000 [==============================] - 2s 81us/step
Test score: 0.04184545138059471
Test accuracy: 0.9875
```

Figure 31: Combination test

Having made some trials for training and testing, I have the best parameter setting for this new network. It is (epoch = 12,batch size = 128,dropout = 0.23,learning rate = 0.01,decay = 1e-6,,momentum = 0.9).
What's more, to make the result more plausible, I make another five trials to test the accuracy in this best parameter setting, which is shown in **Table** 4. What's more, the mean is 0.9862 and variance is 5.71e-06 for this best parameter testing. By this way ,the result is reliable and credible.

Table 4: Accuracy for different five trials in best parameter setting

No.	Accuracy (20000 images)
1	0.9875
2	0.9901
3	0.9843
4	0.9860
5	0.9862

4.3 Discussion

From the last section, we know CNN has a limitation when the background and foreground values are interchanged because of the dependence on the pixel values. To overcome this limitation, in this part, I just double the training set with both original images and negative images while retaining the network structure. By this way, the mixture of original and negative images remove the dependence of background and foreground color. From the result, we see the improvement of robustness of network against both two kind of images.

Reference

[1] Girshick R. Fast R-CNN[J]. Computer Science, 2015.
[2] Gavrilov A, Jordache A, Vasdani M, et al. Convolutional Neural Networks: Estimating Relations in the Ising Model on Overfitting[C]// 2018 IEEE 17th International Conference on Cognitive Informatics and Cognitive Computing (ICCI*CC). 2018.
[3] Brachmann A, Barth E, Redies C. Using CNN Features to Better Understand What Makes Visual Artworks Special.[J]. Frontiers in Psychology, 2017, 8:830-.
[4] Hecht-Nielsen. Theory of the backpropagation neural network[C]// International Joint Conference on Neural Networks. 2002.
[5] Zhao Z H, Yang S P, Zeng-Qiang M A. License Plate Character Recognition Based on Convolutional Neural Network LeNet-5[J]. Journal of System Simulation, 2010, 22(3):638-641.