

University of Southern California
Viterbi School of Engineering
Introduction to Digital Image Processing
2018 Spring EE569

Term Project
Hang Dong
Student ID: 8151308667
donghang@usc.edu
April 25, 2018

Problem 1: CNN Training and Its Application to the MNIST Dataset

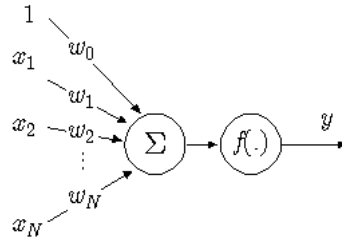
(a) CNN Architecture and Training

1. Basic knowledge

(1) CNN components

The fully connected layer:

Each node in the fully connected layer is connected to all nodes on the previous layer, which combines all the features of previous layer, so the amount of weight parameters at this layer is the most. The network includes a hidden layer, it has a fully connection with input layer and it can be expressed as a function: $\text{output} = f(\sum_i w_i x_i + b_i)$. The function is a activation function, there are some specific widely used ones, like sigmoid function, RELU function. The parameters are weight and bias values between each layer.



Figure_1_1 Fully connection layer

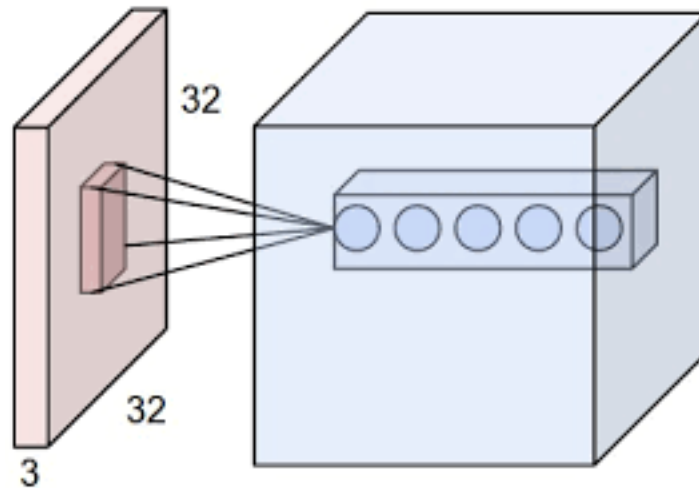
The convolutional layer

There are three convolutional layer in LeNet-5 network, the first one has 6 different kernels (filters) which means it will produce 6 feature maps, the second one has 16 feature maps which are the combination of all or some of 6 maps in previous layer, and the last one has 120 feature maps to give fully connection layer.

The function of convolution layer is to extract various features of the image. It is calculated by a convolution kernel and a sliding window which is called convolution operation. We usually use multi-layer convolution to get a deeper feature map. It can be expressed as a equation:

$$s_{(i,j)} = (X * W)_{(i,j)} + b = \sum_{k=1}^{n_{in}} (X_k * W_k)_{(i,j)} + b$$

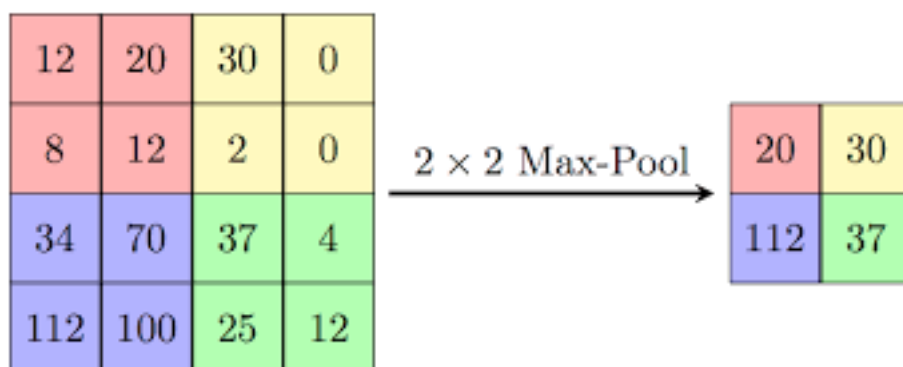
Where n_{in} is the tensor dimension number, X_k is the k th input matrix, W_k is the k th kernel matrix.



Figure_1_2 Convolution

The max pooling:

After obtaining the features of the image through the convolution layer, we can theoretically use these features to train the classifier, but this will face the challenge of huge amount of computation, and it is likely to produce the phenomenon of overfitting. Thus the convolution layer is needed to be followed by pooling, it also called down-sampling. One of pooling methods is called max pooling, it picks the maximum value among the pooling window as the representative value, so the dimension of the input matrix becomes smaller. And we can also change the moving stride to decide how long the center pixel move in every step. In this way, we complete matrix compression.



Figure_1_3 2*2 max-pooling

The activation function

The activation function introduces nonlinear factors to neurons which makes the neural network neural network can be applied to a large number of nonlinear models. It should have non-linear, monotonicity properties and so on.

The first widely used function is Sigmoid function which output value of range is (0,1) which can be used to do two-category classification problem. Sigmoid function is:

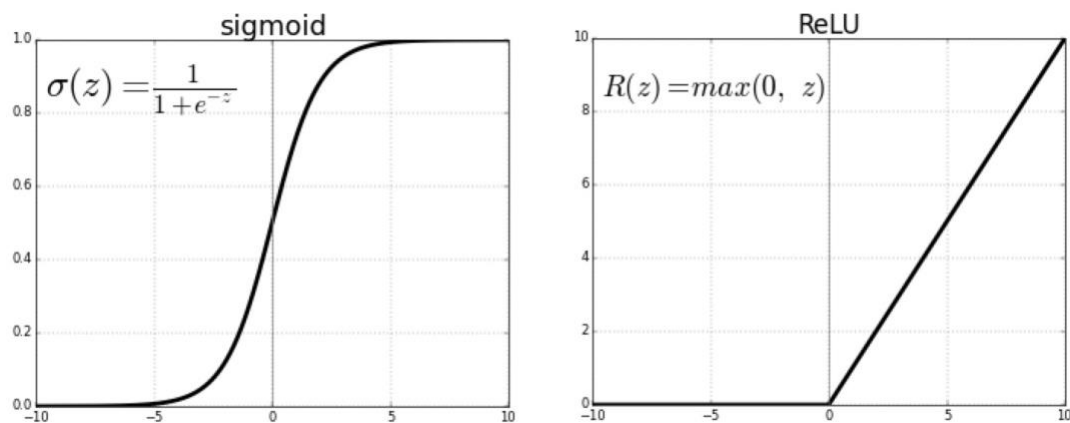
$$f(x) = \frac{1}{1 + e^{-x}}$$

While it has the shortcoming of large amount of calculation (exponent operation and division). What's more, it will occur gradient disperse due to slow transformation speed of sigmoid, making it very slow to train a neural network.

Therefore, we introduce ReLU (Rectified Linear Units) function, which can reduce computation and because its derivative is greater than 0, the gradient will not be smaller, so that we can train a deeper network.

ReLU function is:

$$f(x) = \max(0, x)$$



Figure_2 Activation function of sigmoid and ReLU

Based on figure_2 it's obvious that sigmoid is constantly vary from one to zero which equals to 0.5 when X axis (input) cross zero. Relu is linear ascending when X (input) greater than zero and equals zero when X (input) less than zero.

The soft-max function:

Soft-max can be used for multi-category classification neural network output. Softmax maps a k dimension real value vector (a_1, a_2, \dots) to (b_1, b_2, \dots) where output b_i is a constant range between 0 to 1, and their sum is 1 which means it is actually the probability value. When class number $k = 2$, softmax regression problem degenerates to logistic regression problem.

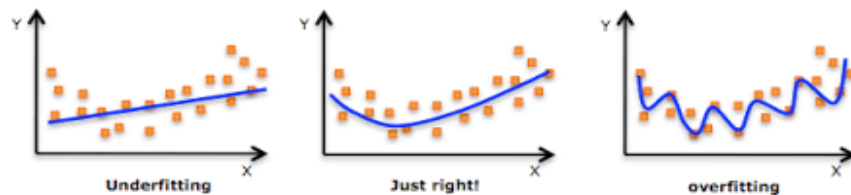
Its function is:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

Overfitting:

It is a phenomenon that if we change the input dataset to this model, there exists huge differences between predicted and the expected value which violates the aim that it can perform well in new samples. It is usually due to the complexity of the model which makes the model is "too good" to learn the given training sample. It is likely to take some of the training sample's own characteristics as the commonality of all potential samples, thus the generalization performance of the model is terrible.

Therefore we can solve it by increasing the amount of data or reducing the number of parameters properly. What is more, we can use regularization method which a regularization term λ is added to the objective equation to not minimize the error between predicted and real value. The objective equation and regularized term are balanced by regularization factors. Furthermore, we can use dropout approach. It makes some activation sample nodes in training procedure to zero randomly with probability p .



CNNs work much better than other traditional methods in many computer vision problems

The traditional neural networks are all connected which will lead to a huge parameter computation, which makes the network difficult to train, while CNN avoids this difficulty by applying local connection and weight sharing which avoids the complexity of data reconstruction in extracting and classifying procedure. Another one of the advantages of CNN compared to the traditional image processing algorithm is that it avoids complex preprocessing such as extraction of artificial features. Because its feature detection layer learns by training data implicitly, the neuron weights are same on the same feature mapping surface, therefore the network can learn in parallel.

Loss function and the classical backpropagation (BP) optimization procedure

We usually use the loss function to evaluate the performance of a neural network. One method to define loss function is calculating the distance between real predicted output \hat{y} and the desired value, the smaller the difference, the better the performance of the neural network. It is called mean square deviation loss function and it can be described as:

$$J(W, b, a, y) = \frac{1}{2} ||a^L - y||_2^2$$

Where a^L and y are feature vector and $||S||_2$ is L_2 norm of S .

We need the back-propagation procedure to update weight and bias parameters in the network by calculating partial derivative $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$. But if we use it combined with Sigmoid activation function, it will result in slow convergence rate in BP algorithm. Therefore we define another definition : cross-entropy loss function.

$$J(W, b, a, y) = -y \cdot \ln a - (1 - y) \cdot \ln(1 - a)$$

Where \cdot is vector inner product. It can overcome the slow updating issue in MSE loss function.

BP:

BP algorithm (back propagation algorithm) is a learning algorithm which is suitable for multilayer neural network under supervision. It is based on gradient descent algorithm. We optimize the parameters of the previous layers in backward direction by iterative algorithm based on minimize loss function until the network get convergence.

For fully connection layer in CNN, we can get the gradient value of W, b based on original BP algorithm, pooling layer does not have W, b so we only need to find out W, b on convolution layer. Because the convolution and pooling layer are not fully connected, we need to do up-sample to return the matrix to the original size before pooling. And if do derivative for the formula in convolution layer, the convolution kernel need be flipped 180 degrees. Therefore, we can derive the derivative of W, b on convolution layer:

$$\frac{\partial J(W, b)}{\partial W^l} = \delta^l * \text{rot180}(a^{l-1})$$

Since b is a vector not a tensor, the usual way is to sum up the elements in each sub-matrix of δ^l to get an error vector as the gradient of b .

$$\frac{\partial J(W, b)}{\partial b^l} = \sum_{u,v} (\delta^l)_{u,v}$$

(b) Train LeNet-5 on MNIST Dataset

1. Abstract and Motivation

Based on the requirements of design, I will train the CNN given flayers parameters as follow:

Convolution layer1: output size is $6*28*28$.

Max pooling layer1: output size is $6*14*14$.

Convolution layer2: output size is $16*10*10$.

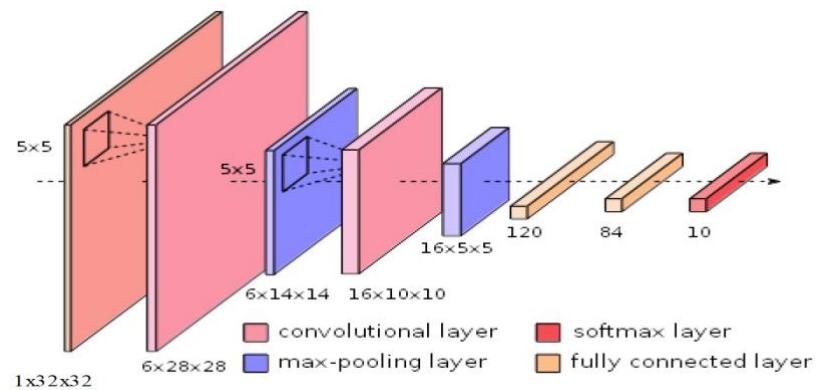
Max pooling layer2: output size is $16*5*5$.

Full connected layer1: output size is 120

Full connected layer2: output size is 84

Full connected layer3: output size is 10

What is more, I will apply the MNIST dataset with 60,000 training images.



Figure_3 LENET5

To build up a LeNet-5 network as the image, I need implement python with tensorflow structure and numpy package.

2. Approach and Procedures

(1). Net structure.

This LeNet5 first convolution layer: input size: $32*32$ (data image);

output size: $6*28*28$

This LeNet5 first max pooling layer: input size: $6*28*28$ (data cube);

output size: $16*14*14$

This LeNet5 third convolution layer: input size: $16*14*14$ (data cube);

output size: $16*10*10$

This LeNet5 first max pooling layer: input size: $16*10*10$ (data cube);

output size: $16*5*5$

This LeNet5 first full connected layer: input size:16*5*5(data cube);

output size: 128

This LeNet5 second full connected layer: input size:128;

output size: 84

This LeNet5 third full connected layer (soft max): input size:84;

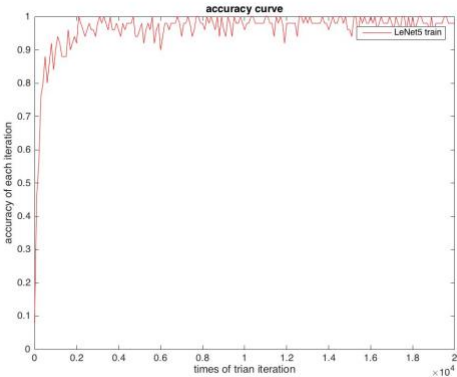
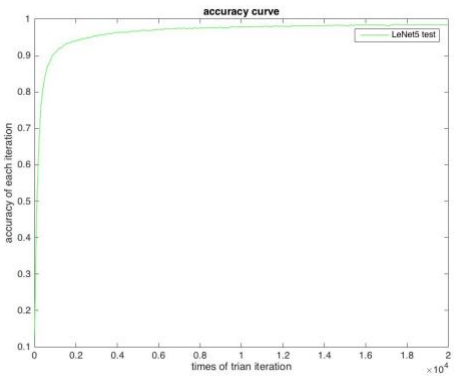
output size: 10

(2). Applied dropout between the second full connected layer and the last to avoid over fitting.

(3). The last full connected layer of this network applied softmax to classify.

3.Experimental Results

#1 result

	
Training curve of accuracy	Testing curve of accuracy
Final accuracy:0.98	Final accuracy:0.9859

Parameters:

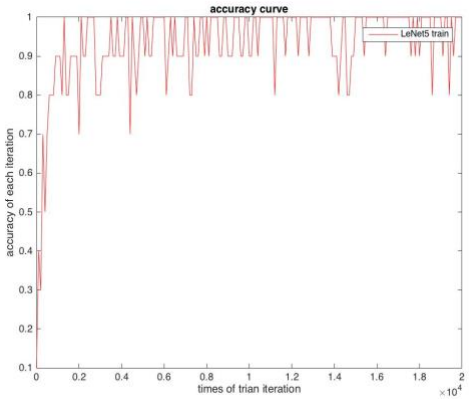
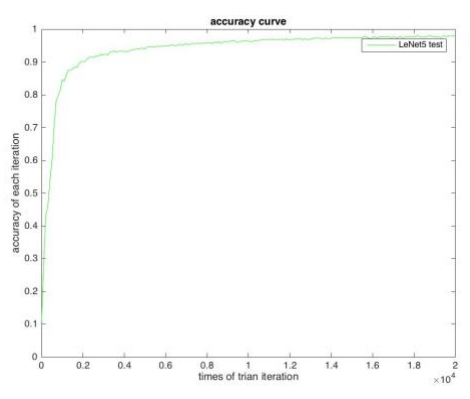
Learning rate: 1e-4

Batch size: 50

weight: Normal (0, $\sigma=0.1$)

Bias: 0.1

#2 result

	
Training curve of accuracy	Testing curve of accuracy
Final accuracy:0.98	Final accuracy:0.9792

Parameters:

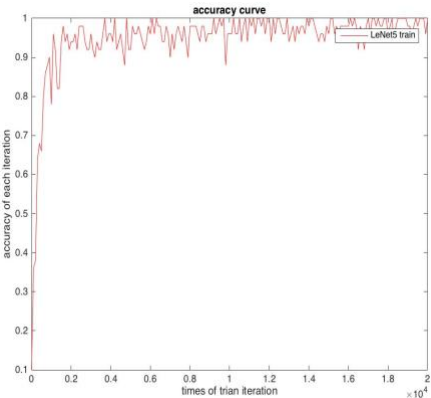
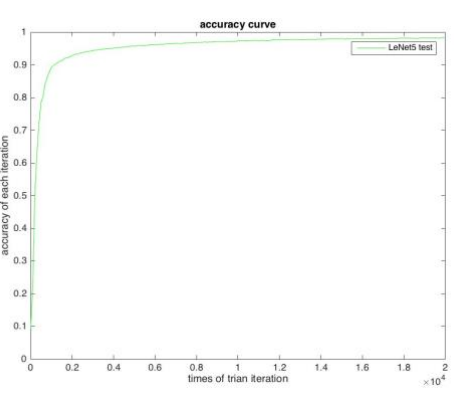
Learning rate: 1e-4

Batch size: 10

weight: Normal (0, $\sigma=0.1$)

Bias: 0.1

#3 result

	
Training curve of accuracy	Testing curve of accuracy
Final accuracy:0.98	Final accuracy:0.983

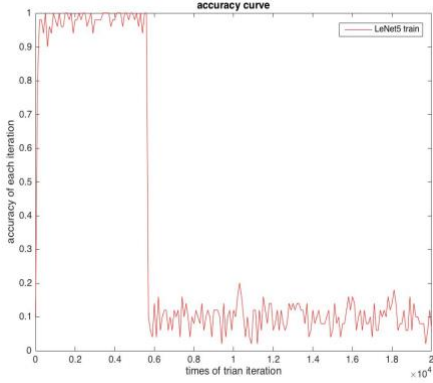
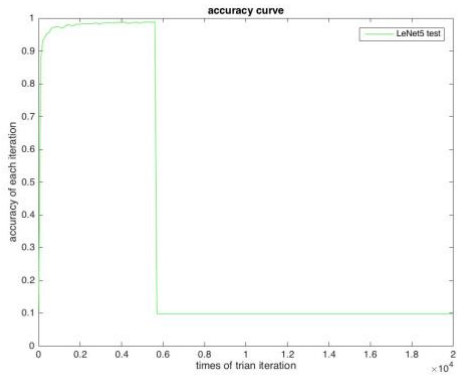
Parameters:

Learning rate: 0.5e-4

Batch size: 50

weight: Normal (0, $\sigma=0.1$)

Bias: 0.1

 <p>The plot shows the training accuracy of a LeNet5 model. The y-axis is 'accuracy of each iteration' from 0 to 1. The x-axis is 'times of trian iteration' from 0 to 2, with a multiplier of $\times 10^4$. The red line starts at 1.0, drops sharply to approximately 0.1 at iteration 0.6, and then fluctuates between 0.1 and 0.2 for the remainder of the training.</p>	 <p>The plot shows the testing accuracy of a LeNet5 model. The y-axis is 'accuracy of each iteration' from 0 to 1. The x-axis is 'times of trian iteration' from 0 to 2, with a multiplier of $\times 10^4$. The green line starts at 1.0, drops sharply to approximately 0.1 at iteration 0.6, and then remains constant at 0.1 for the remainder of the training.</p>
Training curve of accuracy	Testing curve of accuracy
Final accuracy:0.102	Final accuracy:0.083

Parameters:

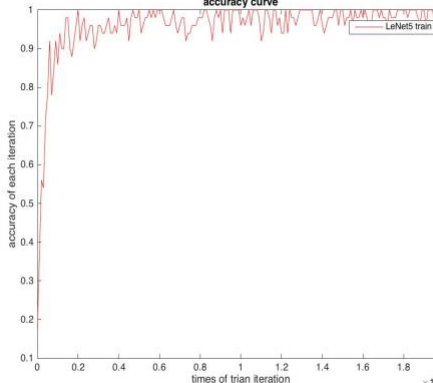
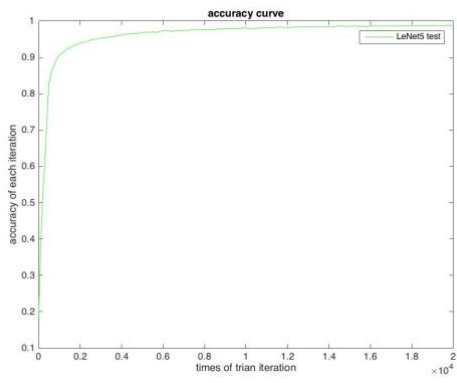
Learning rate: 2e-4

Batch size: 50

weight: Normal (0, $\sigma=0.1$)

Bias: 0.1

#4 result

 <p>The plot shows the training accuracy of a LeNet5 model. The y-axis is 'accuracy of each iteration' from 0.1 to 1. The x-axis is 'times of trian iteration' from 0 to 2, with a multiplier of $\times 10^4$. The red line starts at 0.1, rises sharply to approximately 0.9 at iteration 0.2, and then fluctuates between 0.9 and 1.0 for the remainder of the training.</p>	 <p>The plot shows the testing accuracy of a LeNet5 model. The y-axis is 'accuracy of each iteration' from 0.1 to 1. The x-axis is 'times of trian iteration' from 0 to 2, with a multiplier of $\times 10^4$. The green line starts at 0.1, rises sharply to approximately 0.9 at iteration 0.2, and then remains constant at 0.9 for the remainder of the training.</p>
Training curve of accuracy	Testing curve of accuracy
Final accuracy:0.97	Final accuracy:0.9831

Parameters:

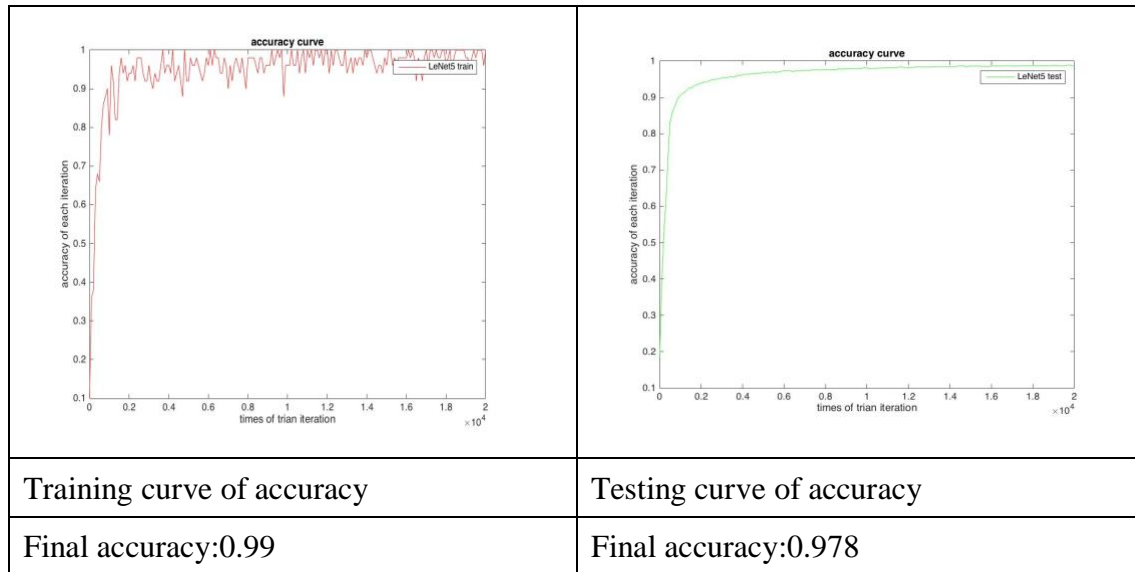
Learning rate: 1e-4

Batch size: 50

weight: Normal (0, $\sigma=0.3$)

Bias: 0.1

#5 result



Parameters:

Learning rate: 1e-4

Batch size: 50

weight: Normal (0, $\sigma=0.1$)

Bias: 0.25

According to these 5 different results, I can identify the influence of different parameters. Assumed #1 is the basic simulation to compare with other results.

Result 1 vs Result 2(changing batch size)

I changed batch size to 10, according to the training accuracy curve its becomes more unstable and hard to get convergence. Finally, the accuracy is lower than result 1.

Result 1 vs Result 3 (changing learning rate)

I changed learning rate to 0.5e-4 and 2e-4. According to the training accuracy curve and testing accuracy curve it is obvious when learning rate is so greater it will miss the optimization result and the learning result will get worse and worse final test accuracy.

Result 1 vs Result 4(changing weight)

I changed weight standard parameter $\sigma=0.3$, Its similar to the result 1. They have similar training and testing curves.

Result 1 vs Result 5(changing bias)

By changing bias from 0.1 to 0.25, the final accuracy gets worse than original result 1. That's resulted by bias value is constant, if we give more attention to this part its obvious the full network will tend to be more linear than randomly.

4. Discussion

(1) In my implemented LeNet5 network, parameters as follow:

#1. Filter weight: I initialed weight equals 0.1. To initial the filter weight, I use random way by applying normal distribution. Network's training results can do improvement after different iterations instead of repeat same computations.

#2. Learning rate: I initialed the learning rate equals to 1E-4, which after tested is a good learning rate to find the global optimization. If learning rate is too large then the global optimization will miss out, otherwise the network can not find the optimization result efficiently. After times of times test I chose 1E-4 as learning rate.

#3. Decay: Decay is used to raising the update speed of weight and bias parameters.

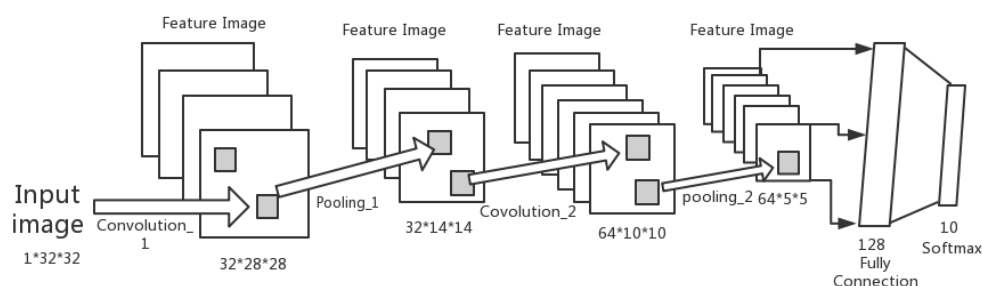
#4. Bias: Usually set up to find better aim(optimization) initial as a constant value=0.1, in the network I need plus bias to each layer.

#5. Batch size: Influence the performance stable of each iteration and the total iteration times to find the optimization result. Too big size will make the computation very complexity and too small will result more iterations and unstable in each iteration.

(c) Improve the LeNet-5 for MINST dataset

1. Abstract and Motivation

Train the CNN given below using the 60,000 training images from the MNIST dataset.



Figure_4 improve CNN

2. Approach and Procedures

(1). Net structure.

This LeNet5 first convolution layer: input size:32*32(data image);

output size: 32*28*28

This LeNet5 first max pooling layer: input size: 32*28*28(data cube);

output size: 32*14*14

This LeNet5 second convolution layer: input size: 32*14*14(data cube);

output size: 64*10*10

This LeNet5 second max pooling layer: input size: 64*10*10(data cube);

output size: 64*5*5

This LeNet5 first full connected layer: input size: 64*5*5(data cube);

output size: 128

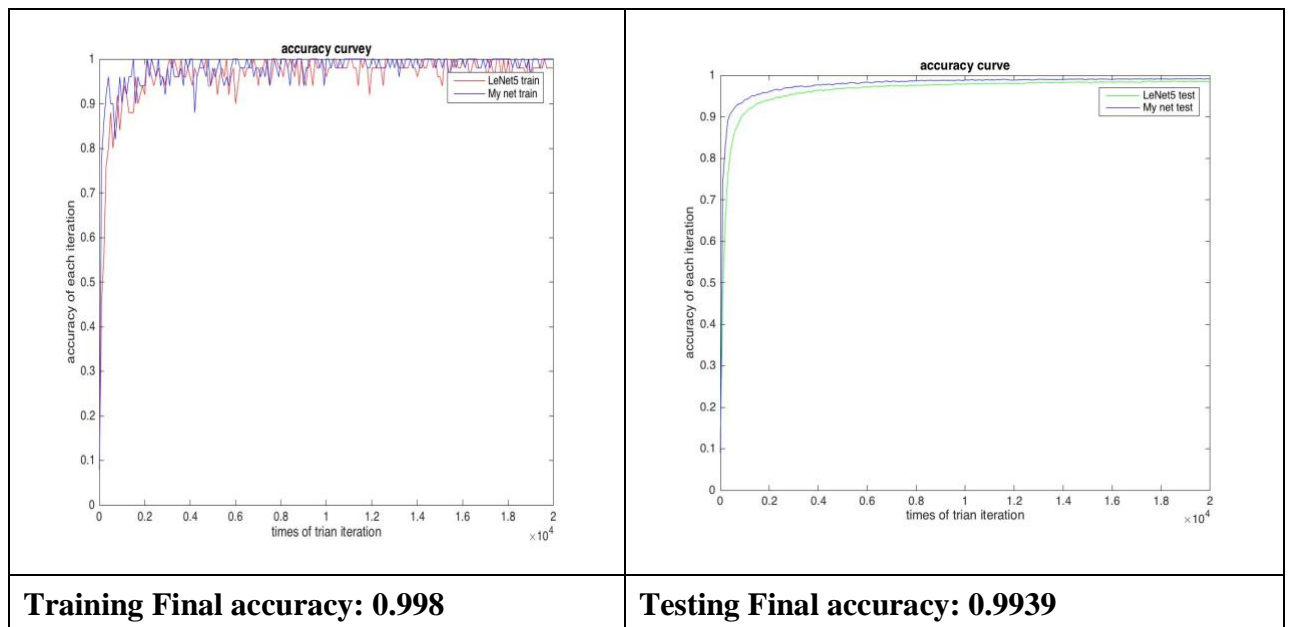
This LeNet5 second full connected layer: input size:128;

output size: 10

(2). Applied dropout between the first full connected layer and the last to avoid over fitting.

(3). The last full connected layer of this network applied softmax to classify.

3.Experimental Results



4.Discussion

My network has similar structure as LeNet5 network but modified some parameters and delete one full connected layers to simplify it. According to test performance, my network is obviously get the better accuracy which accuracy equals to 0.9939, my

network's accuracy curve is above the standard LeNet5 curve. So that is mean my network is better than standard LeNet5 when deal with MNIST data size. However, due to CNN is not a data driven method, its hard to say my network can has better results when deal with other kinds of data.

Problem 2: Saak Transform and Its Application to the MNIST Dataset

(a) Comparison between the Saak Transform and the CNN architecture

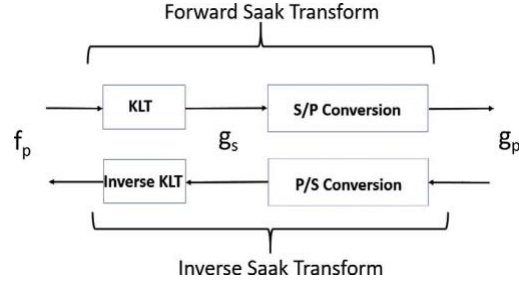
1. Basic knowledge

There are some different perspectives to learn the internal principles in CNN. The signal viewpoint is a novel one, and it come up with a new module REctified COrrrelations on a Sphere (RECOS) to explain the non-linearity in CNN network. As we all know, applying labeled (supervised way) training data and backpropagation to optimize weight parameters. In RECOS model, using weight parameters as anchor vectors in order to calculate the similarity among themselves and input vectors, anchor vector projects kindred input data to the same output area and that is why CNN can identify different objects effectively.

Based on above analysis, we can also observe some disadvantages in RECOS transform: the rectification loss and the approximation loss. The rectification loss is caused by setting negative values to 0 directly in ReLU function which makes the signal representation less effective. The approximation loss is caused by kernel number is scant and further worse, the reverse RECOS is not straightforward because of the parameters we pick. In order to solve questions above, we introduce Subspace approximation with augmented kernels (Saak) transform. Subspace approximation uses the KLT transform dimension reduction approach to pick a set of eigenvectors in the covariance matrix as transformation kernels in order to deal with the approximation loss trouble. Kernels augmentation is to deal with rectification loss trouble. It includes each kernel with both '+' and '-' component in order to make one of them via the non-linear activation function. The procedure can be described as sign-to-position (S/P) conversion.

The whole procedure are three steps: calculate Saak coefficient, Saak coefficients

selection and dimension reduction and apply classifier to extracted features. The first step can be further divided into KLT and S/P conversion sub-step, as figure_5 showing:



Figure_5 S/P conversion

Firstly, get several patches sized 2×2 by departing the original input image and remove these patches' mean and computed the variance matrix of each patch (which also called KLT transform). we can select the largest server orthonormal kernels which is known as truncated KLT to reduce time complexity and use less memory resource. Then we increase each kernel by adding opposite one, $a_{2k-1} = b_k, a_{2k} = -b_k$ which can be compressed as S/P conversion. To be specifically, the input image size is $32 \times 32 \times 1$ (MNIST dataset standard size), Do multiple Saak transform repeatedly until we get the root 1×1 . After we obtain raw Saak coefficient, we can further use PCA approach to decrease kernel dimension. Here we also introduce F-test score method. We calculate F-test score for each kernel and select the largest several ones with larger distinguishable power. The calculation equation is:

$$F = \frac{\text{between - group variability(BGV)}}{\text{within - group variablity(WGV)}}$$

$$BGV = \sum_{c=1}^C n_c (\bar{S}_c - \bar{S})^2 / C - 1$$

$$WGV = \sum_{c=1}^C \sum_{d=1}^{n_c} (S_{c,d} - \bar{S}_c)^2 / T - C$$

Finally we can input these selected features to different classifiers (SVM, RF) to train classifier and compare the classification results.

(b) Application of Saak transform to MNIST dataset

1. Abstract and Motivation

Saak transform is a new method that just invented recently. Saak is totally different from CNN that it is based on data. In other words, Saak transform has better adaptive property than traditional CNN network method. Firstly, Saak transform calculate the saak coefficients of input data. Then use Saak models to calculate the feature coefficients repeatedly to get the final feature has size of $1*1*N$. Then using **Ftest** and **PCA** to reduce the features dimensions to 32, 64 and 128 and then use SVM/RF to classify it.

2.Approach and Procedures

step1: Calculate Saak coefficients using non-overlapping patch region with spatial size $2*2$. Do this step repeatedly until the feature size become $1*1*N$.

step2: Using Ftest and PCA to reduce Saak coefficient' dimensions. To test the performance, I use feature dimension of PCA equals to 32, 64 and 128.

step3: Separately using SVM and Random Forest to classifying and test accuracy.

3.Experimental Results

Accuracy (different PCA components)	32	64	128
SVM	0.981	0.984	0.983
Random Forest	0.973	0.971	0.972

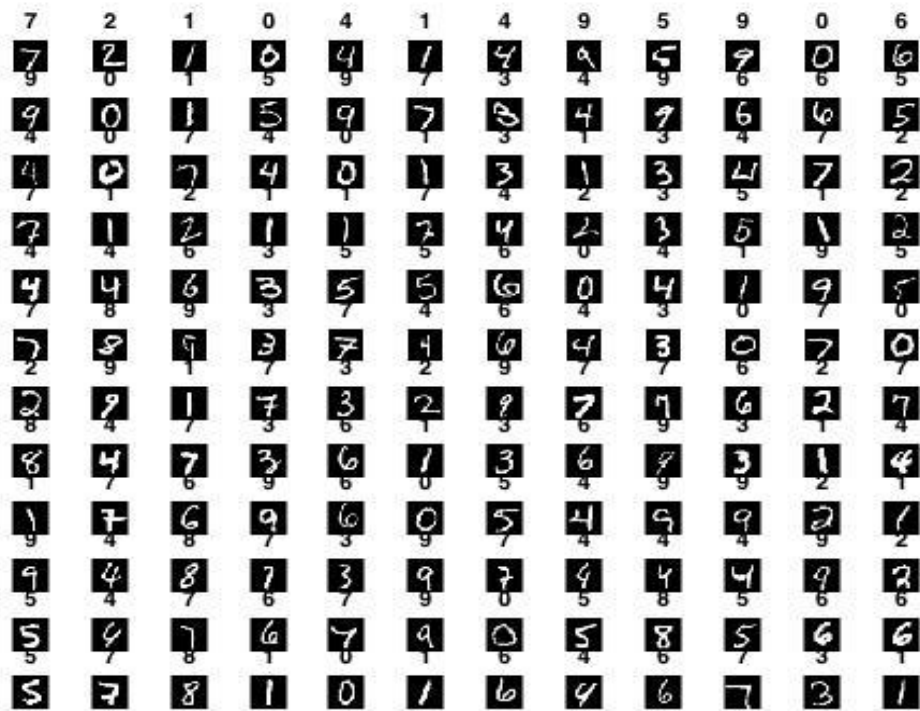
According to the result sheet SVM can get better result than RF method. With different final PCA components separately equals to 32, 64 and 128, the accuracy of higher dimensions will get better and the result of 64 and 128 are similar.

4.Discussion

According to my experiment results, the results of Saak transform and LeNet5 CNN network are similar but CNN can get higher accuracy with good network structure. What is more, in Saak transform result SVM's performance is better than RF's results.

(c) Error Analysis

error



Figure_6 MNIST

According to the test classify results, the mostly wrong classify results are the data parts as follow:

‘5’ and ‘3’;

‘1’ and ‘7’;

‘4’ and ‘9’

According to the results of Saak transform and CNN, the percentages of errors are the same as follow:

According to my test the best result can get by set Parameter: **PCA=64**

Percentage(%)	percentages of errors are same	percentages of errors are different
Saak(SVM) & CNN	2.3%	1%
Saak(RF) & CNN	3.2%	1.4%

According to my observations of the error cases of saak and cnn results, its obvious that the error cases have more similar cases than difference, which mean these two methods’ feature are similar. What’s more, it is obvious that SVM has better results than RF due its accuracies are greater than RF’s all time.

Explanations and propose

According to my observation, I observe that the results of Saak transform and LeNet5 CNN network are similar but CNN can get higher accuracy with good network structure. According to this report problem 1, my improved network has better accuracy than standard LeNet5 of 0.8%. What is more, in Saak transform result SVM's performance is better than RF's results. Averagely the results of SVM's accuracy better than RF of 1.1%.

To improve the performance of CNN, I think we can try to add more convolution and max-pooling layers because convolution layers can help blurring the original image to ignore the useless details and retain the main features. The network architecture and parameter adjusting greatly affect the training and performance of CNN. We can improve the performance of CNN network from these two aspects. We can use convolution layer to reduce the parameter number in fully connection layer which makes the learning process easier. Furthermore, we can use more powerful regularization techniques, especially dropout and convolution, to reduce overfitting problem.

To improve the performance of Saak, I think we can try to apply convolution layer firstly and then do saak transform. With the help of convolution layers, which helps the algorithm to get efficient features of the original image efficiently. After that I do saak transform and feature dimension reduction.

Reference

- [1] <http://yann.lecun.com/exdb/lenet/index.html>
- [2] C.-C. J. Kuo, "CNN as Guided Multi-layer RECOs Transform"
- [3] <https://github.com/davidsonic/Saak-Transform>
- [4] slides in discussion given by Haiqiang Wang, Ye Wang, Yueru Chen.