# PROJECT 3\_Neural Network

目录

[**PROJECT 3\_Neural Network 1**](#_Toc469243893)

[**TasK 1： 1**](#_Toc469243894)

[**Answer: 2**](#_Toc469243895)

[**Comparison: 2**](#_Toc469243896)

[**TasK 2: 2**](#_Toc469243897)

[**Answer: 2**](#_Toc469243898)

[**Comparison: 5**](#_Toc469243899)

[**Task 3: 6**](#_Toc469243900)

[**Answer: 6**](#_Toc469243901)

[**Task 4: 7**](#_Toc469243902)

[**Answer: 7**](#_Toc469243903)

[**Task 5: 9**](#_Toc469243904)

[**Answer: 9**](#_Toc469243905)

[**Task 6: 13**](#_Toc469243906)

[**Answer: 13**](#_Toc469243907)

[**Task 7: 15**](#_Toc469243908)

[**Answer: 15**](#_Toc469243909)

[**Task 8 18**](#_Toc469243910)

[**Answer: 18**](#_Toc469243911)

[**Task 9: 20**](#_Toc469243912)

[**Answer: 20**](#_Toc469243913)

[**Task 10: 23**](#_Toc469243914)

[**Answer: 23**](#_Toc469243915)

[**Task \* 29**](#_Toc469243916)

[**Conclusion: 29**](#_Toc469243917)

[**Note : 31**](#_Toc469243918)

[**Refenrence 31**](#_Toc469243919)

## TasK 1：

【Change the network structure : the vector nHidden specifies the number of hidden units in each layer.】

### Answer:

【% Choose network structure nHidden = [10];】

We find [n] means a one layer nerual network,[n1,n2,n3,…,n7]means a 7 layer nerual network. The dimension of the vector of ’nHidden’ equals to the number of nerual network’s layer.

【we chose [7],[100],[7,7,7],[100,100]】

We get the test error with final model equals to 0.540000 , 0.222000, 0.590000, 0.266000 respectively.

So it seems that the when the vector nHidden equals to [100] ,we gets the best answer.

### Comparison:

|  |  |
| --- | --- |
| Coming back from the afterwards.  After modifing the paramters again and again. | |
| We find the best nHidden=[100] | the other best nHidden=[120,60,30] |
| Test error=  Valid error= | Test error=  Valid error= |

## TasK 2:

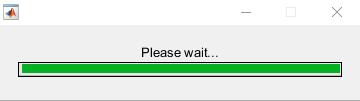
【change the training procedure by modifying the sequence of step-sizes or using different step for different variables.The momentum uses the update as the question2 shows.】

### Answer:

【we change the nHidden to its intial value, nHidden = [10];in this way it is more convenient for us to compare the result.】

Since the time for training the model is a crucial part for the efficency of a nerual network. I decide to add the “waitbar ” to show whether a code is feasible.

|  |
| --- |
| h = waitbar(0,'Please wait...'); for i=1:1000   % computation here %  waitbar(i/1000,h)   end |



【Than firstly modify the code in order to realize the momentum part】

In order to record the .I introduce a new variable J, and the code shows as below

|  |
| --- |
| % Train with stochastic gradient  funObj = @(w,i)MLPclassificationLoss(w,X(i,:),yExpanded(i,:),nHidden,nLabels);  iter = 1;  if mod(iter-1,round(maxIter/20)) == 0  yhat = MLPclassificationPredict(w,Xvalid,nHidden,nLabels);  fprintf('Training iteration = %d, validation error = %f\n',iter-1,sum(yhat~=yvalid)/t);  end  i = ceil(rand\*n);  [f,g] = funObj(w,i);  w = w - stepSize\*g;  J=[w];  b=0.9;  h = waitbar(0,'Please wait...');  for iter = 2:maxIter  if mod(iter-1,round(maxIter/20)) == 0  yhat = MLPclassificationPredict(w,Xvalid,nHidden,nLabels);  fprintf('Training iteration = %d, validation error = %f\n',iter-1,sum(yhat~=yvalid)/t);  end  i = ceil(rand\*n);  [f,g] = funObj(w,i);  w= w - stepSize\*g+b\*(w-J);  J=[w];  waitbar((iter-1)/(maxIter-1),h)  end |

|  |
| --- |
| **There is also another way to intialize the weight function**  % Train with stochastic gradient  maxIter = 100000;  weights=zeros(nParams,2);  weights(:,1)=w; %another way to compute the new w!  b=0.9;  stepSize =2e-3;  funObj = @(w,i)MLPclassificationLoss3(w,X(i,:),yExpanded(i,:),nHidden,nLabels);  i = ceil(rand\*n);  [f,g] = funObj(w,i);  % another way to renew the w!  if iter ==1  w=w-stepSize\*g;  weights(:,2)=w;  else  w=w-stepSize\*g+b\*(w-weights(:,2));  weights(:,1)=weights(:,2);  weights(:,2)=w;  end |

And than we have the result as following shows.

Test error with final model = 0.214000 when nHidden equals to 100. A little better than the value in the first task.

we also find the validation error doesn’t rigidly decrease when the iteration number increases.

It has a fluctuation when the iteration number is close to 50000.

|  |
| --- |
| Training iteration = 50000, validation error = 0.230400  Training iteration = 55000, validation error = 0.238400  Training iteration = 60000, validation error = 0.240000  Training iteration = 65000, validation error = 0.259800  Training iteration = 70000, validation error = 0.232000  Training iteration = 75000, validation error = 0.228200 |

【than I want to modify the step-size】

I think the stepSize should change with the validation error. When the validation error is big, the stepSize should be much larger as a coarse control [粗调] in order to change the w more rapidly. While when the validation error is small,the stepSize should be much smaller as a delicating adjusting [细调] .

|  |
| --- |
| fprintf('Training iteration = %d, validation error = %f\n',iter-1,sum(yhat~=yvalid)/t);  if sum(yhat~=yvalid)/t >= 0.8  stepSize=0.01  end  if sum(yhat~=yvalid)/t >= 0.5 && sum(yhat~=yvalid)/t < 0.8  stepSize=0.005  end  if sum(yhat~=yvalid)/t >= 0.3 && sum(yhat~=yvalid)/t < 0.5  stepSize=0.001  end  if sum(yhat~=yvalid)/t >= 0 && sum(yhat~=yvalid)/t < 0.3  stepSize=0.0003  end |

With this modify, when nHidden=[10] the previous Test error with final model = 0.502000,but now the Test error with final model = 0.458000. which is better than before.

【combine with these two modify】

We finally get the best test error when nHidden=[100] the Test error with final model = 0.206000, A little better than before.

### Comparison:

|  |
| --- |
| Coming back from the afterwards.  After modifing the paramters again and again. |
| We find the best parameter[which contains b & stepsize]show as bellow |

|  |
| --- |
| nHidden=[100] |
| if iter>maxIter/5  stepSize=1e-3;  b=0.9;  end  if iter>maxIter\*2/5  stepSize=5e-4;  b=0.7;  end  if iter>maxIter\*3/5  stepSize=5e-5;  b=0.2;  end  if iter>maxIter\*4/5  stepSize=1e-5;  b=0.1;  end  end |

|  |
| --- |
| nHidden=[120,60,30] |
| if iter>maxIter/5  stepSize=2e-3;  b=0.9;  end  if iter>maxIter\*2/5  stepSize=2e-3;  b=0.75;  end  if iter>maxIter\*3/5  stepSize=1e-3;  b=0.5;  end  if iter>maxIter\*4/5  stepSize=5e-4;  b=0.35;  end  end |

b and the stepSize both change with the maxlter.

## Task 3:

【you could vectorize evaluating the loss function allow you to do more training iterations in a reasonable time.】

### Answer:

In order to train the model more quickly we need to change the iteration calculation into the matrix calculation. So we modify the code as follow.

First we add code.1 for calculating time.

|  |
| --- |
| tic  …  t=toc;  fprintf('time= %f\n',t); |

Next we add code.2 for matrix calculation.

We modify the MLPclassficationLoss function

|  |
| --- |
| ①  %for c = 1:nLabels  %gOutput(:,c) = gOutput(:,c) + err(c)\*fp{end}';  %end  gOutput = (fp{end}'\*err);  ②  %for c = 1:nLabels  % backprop(c,:) = err(c)\*(sech(ip{end}).^2.\*outputWeights(:,c)');  % gHidden{end} = gHidden{end} + fp{end-1}'\*backprop(c,:);  backprop = err\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  gHidden{end} = gHidden{end} + fp{end-1}'\*backprop;  ③  % Input Weights  % for c = 1:nLabels  % gInput = gInput + err(c)\*X(i,:)'\*(sech(ip{end}).^2.\*outputWeights(:,c)');  % end  gInput = gInput + X(i,:)'\*err\*(repmat(sech(ip{end}),nLabels,1).^2.\*outputWeights'); |

In this way we get the result of the time= 175.503440,when nHidden=[100], which is pretty better than the previous time=361.288116

## Task 4:

【Add L2 regularization(L1 regularization ) For neural networks this is called weight decay. An alternate form of regularization of regularization that is sometimes used is early stopping.】

### Answer:

As we see ,there are two tasks first is L2 regularization .the second is early stopping.



The first task we should add a L2 reugularization like  and we have the gradient like  , so we intend to add this gradient and the regularization respectively to the new gradient and the loss function (since the loss function doesn’t use for renew the weight so here we only change the gradient by adding the L2 )

So the code as follow part:

|  |
| --- |
| if nargout > 1  err = 2\*relativeErr;  % Output Weights  %for c = 1:nLabels  % gOutput(:,c) = gOutput(:,c) + err(c)\*fp{end}';  %end  gOutput = (fp{end}'\*err)+lambda\*(outputWeights);  if length(nHidden) > 1  % Last Layer of Hidden Weights  clear backprop  backprop = err\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  gHidden{end} = gHidden{end} + fp{end-1}'\*backprop+lambda\*hiddenWeights{length(nHidden)-1};  backprop = sum(backprop,1);  % Other Hidden Layers  for h = length(nHidden)-2:-1:1  backprop = (backprop\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  gHidden{h} = gHidden{h} + fp{h}'\*backprop+lambda\*(hiddenWeights{h});  end  % Input Weights  backprop = (backprop\*hiddenWeights{1}').\*sech(ip{1}).^2;  gInput = gInput + X(i,:)'\*backprop+ lambda\*(inputWeights);  else  % Input Weights  % for c = 1:nLabels  % gInput = gInput + err(c)\*X(i,:)'\*(sech(ip{end}).^2.\*outputWeights(:,c)');  % end  gInput = X(i,:)'\*err\*(repmat(sech(ip{end}),nLabels,1).^2.\*outputWeights')+lambda\*(inputWeights);  end |

Then we have the result when nHidden=[20]

|  |
| --- |
| lambda=0.2  Training iteration = 85000, validation error = 0.167200  Training iteration = 90000, validation error = 0.170200  Training iteration = 95000, validation error = 0.181600  Test error with final model = 0.137000  time= 113.337589 |

But when we choose the nHidden=[20 20] these two layer, than we have to chose a different lambda = 0.01 than we have the result:

|  |
| --- |
| lambda=0.01  Training iteration = 85000, validation error = 0.320600  Training iteration = 90000, validation error = 0.297000  Training iteration = 95000, validation error = 0.270000  Test error with final model = 0.232000  time= 196.199335 |

With different lambda the weight decay gives a better answer.

Next the early stopping’s code

|  |
| --- |
| if mod(iter-1,round(maxIter/20)) == 0  yhat = MLPclassificationPredict(w,Xvalid,nHidden,nLabels);  A= sum(yhat~=yvalid)/t;  fprintf('Training iteration = %d, validation error = %f\n',iter-1,A);  end  i = ceil(rand\*n);  [f,g] = funObj(w,i);  w = w - stepSize\*g;  J=[w];  b=0.9;  h = waitbar(0,'Please wait...');  for iter = 2:maxIter  if mod(iter-1,round(maxIter/20)) == 0  yhat = MLPclassificationPredict(w,Xvalid,nHidden,nLabels);  fprintf('Training iteration = %d, validation error = %f\n',iter-1,sum(yhat~=yvalid)/t);  end  if sum(yhat~=yvalid)/t >A  break  else A=sum(yhat~=yvalid)/t;  end  i = ceil(rand\*n);  [f,g] = funObj(w,i);  w= w - stepSize\*g+b\*(w-J);  J=[w];  waitbar((iter-1)/(maxIter-1),h)  end |

We retry the condition that nHidden=[20];

We break the iteration at 30000 times.and got the best test error ever since.

|  |
| --- |
| Training iteration = 30000, validation error = 0.126000  Test error with final model = 0.111000 |

## Task 5:

【Instead of using the squared error ,use a softmax(multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class.】

### Answer:

Softmax turns out to be one of 2 commonly seen classifiers.(another is svm), indeed softmax is binary Logistic Regression classifier’s generalization to the multiple class.

Softmax has a advantage of interpretation. It produces: the correct class always have a higher probability and the incorrect classes always a lower probability and the loss would always get better.

In this task ,we want to use the softmax function to take the place of the squred error loss function.

Since the softmax function is like 

In which the expression of 

So at first we write the function of choice the right 

Since the function [y] = linearInd2Binary(ind,nLabels)，it means we modify the result function to let the not real labels equals to -1,while the true label equal to 1. So only the real labels equal to 1.

So the code we write to construct the softmax loss function as following:

|  |
| --- |
| %softmax  %choice the right yhat  for m=1:10  if y(m)==1  break  end  end  k=exp(yhat(m));  %compute Loss function  p=0;  for j=1:10  p=exp(yhat(j))+p;  end  probability=exp(yhat)/p;  f=f -log(k/p); |

Than since the loss function has changed so the gradient change also.





Through simplification we have the result that:





Notice that the denominator is a value and the numerator is a [1\*10]vector and minus a [n\*10]matrix

While we want to get the derivative of a matrix. And each derivative of the matrix have nothing to do with other column of the matrix.  
[for example : the  has nothing to do with z1,z3…zn it only has relationship with z2.]



So we have a the simplify of the derivative that



And the code is as follow

|  |
| --- |
| gOutput = fp{end}'\*exp(yhat)/p+lambda\*(outputWeights);  gOutput(:,m) = gOutput(:,m)-fp{end}'; |

While the loss function is change and the coefficient is changed.Although the way of gradient of layer is not change. we need to modify it by change the coefficient.

So the other changed code is as follow.

|  |
| --- |
| if nargout > 1  %output layer  gOutput = fp{end}'\*exp(yhat)/p+lambda\*(outputWeights);  gOutput(:,m) = gOutput(:,m)-fp{end}';  if length(nHidden) > 1  % Last Layer of Hidden Weights  clear backprop  clear backprop1%/p  backprop = exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights')/p;  backprop1= sech(ip{end}).^2.\*outputWeights(:,m)';  gHidden{end} = gHidden{end} + fp{end-1}'\*backprop-fp{end-1}'\*backprop1+lambda\*hiddenWeights{length(nHidden)-1};  backprop = sum(backprop,1);  % Other Hidden Layers  for h = length(nHidden)-2:-1:1  backprop = (backprop\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  backprop1 = (backprop1\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  gHidden{h} = gHidden{h} +(fp{h}'\*backprop-fp{h}'\*backprop1)/p+lambda\*(hiddenWeights{h});  end  % Input Weights  backprop = (backprop\*hiddenWeights{1}').\*sech(ip{1}).^2;  backprop1 = (backprop1\*hiddenWeights{1}').\*sech(ip{1}).^2;  gInput = gInput + (X(i,:)'\*backprop- X(i,:)'\*backprop1)/p+ lambda\*(inputWeights);  else  %nargout = 1  mya= X(i,:)'\*exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  myq =X(i,:)'\*p\*(sech(ip{end}).^2.\*outputWeights(:,m)');  gInput = gInput+(mya-myq)/p+lambda\*(inputWeights);  end |

In this task 5 we include the optimization of the task 4 , and we have the result when nHidden=[20] the stepsize=1\*10^(-3) and when nHidden=[20 20] the stepsize=2\*10^(-3).

|  |
| --- |
| ①  validation error = 0.916000--0.764600--0.645800--0.553600--0.489400--0.430600--0.388200--0.353200--0.316200--0.286400--0.252400--0.224400--0.202800--0.182600--0.164200--0.148600--0.132400--0.122200  Training iteration = 90000, validation error = 0.112200  Training iteration = 95000, validation error = 0.102600  Test error with final model = 0.086000  ②  validation error = 0.905600--0.701200-- 0.570000-- 0.504800-- 0.458600-- 0.420800-- 0.408600-- 0.373800-- 0.346400--0.317200--0.285400--0.253400--0.220800--0.193200--0.165000--0.147800--0.136200-- 0.118400  Training iteration = 90000, validation error = 0.113600  Training iteration = 95000, validation error = 0.1046006  Test error with final model = 0.088000 |

We can see the result is much better than before.

## Task 6:

【instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant.】

### Answer:

That means in order to make the bias variable. We arenot required to add a different bias variable everytime in each layer. Instead we can add a 1 to the x input.

For the first input layer:

In other word, the x matrix change from 5000\*256 to 5000\*257 by adding Than the first layer weight matrix change from nhidden(1)=N1\*256 to N1\*257 the last term of the weight is this is the bias.

For the nhidden layers:

Instead of adding a 1 term in each layer’s input, we notice that the weight term is derived by random selected. So we change the last layer’s input term to be 1,this equals to decide the last weight which makes the

.

This makes the(L+1) layer has the bias term.

|  |
| --- |
| for i = 1:nInstances  ip{1} = X(i,:)\*inputWeights;  fp{1} = tanh(ip{1});  fp{1}(end)=1;  for h = 2:length(nHidden)  ip{h} = fp{h-1}\*hiddenWeights{h-1};  fp{h} = tanh(ip{h});  fp{h}(end)=1;  end  yhat = fp{end}\*outputWeights; |

Remark: since we have the bias derivatives without the regularization term.



So in this part we also need to modify the derivative as well. Indeed, note that biases do not have the same effect as the weights. Unlike the weights, they do not decide the structure of the curve. So that it has no relationship with overfitting. It also don’t control the strength of influence of an input dimension.

Therefore, it is common to only regularize the weights but not the biases.

|  |
| --- |
| if nargout > 1  %output layer  gOutput = fp{end}'\*exp(yhat)/p+lambda\*(outputWeights);  gOutput(:,m) = gOutput(:,m)-fp{end}';  if length(nHidden) > 1  % Last Layer of Hidden Weights  clear backprop  clear backprop1%/p  backprop = exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights')/p;  backprop1= sech(ip{end}).^2.\*outputWeights(:,m)';  gHidden{end}=gHidden{end}+fp{end-1}'\*backprop-fp{end-1}'\*backprop1;  add=[];  add= hiddenWeights{length(nHidden)-1};  [ap,aq]=size(add);  add(:,aq)=0;  gHidden{end}=gHidden{end}+lambda\*add;  backprop = sum(backprop,1);  % Other Hidden Layers  for h = length(nHidden)-2:-1:1  backprop = (backprop\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2; backprop1 = (backprop1\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  gHidden{h}=gHidden{h}+(fp{h}'\*backprop-fp{h}'\*backprop1)/p;  add=[];  add= hiddenWeights{h};  [ap,aq]=size(add);  add(:,aq)=0;  gHidden{h}=gHidden{h}+lambda\*add;  end  % Input Weights  backprop = (backprop\*hiddenWeights{1}').\*sech(ip{1}).^2;  backprop1 = (backprop1\*hiddenWeights{1}').\*sech(ip{1}).^2;  gInput = gInput + (X(i,:)'\*backprop- X(i,:)'\*backprop1)/p;  add=[];  add= inputWeights;  [ap,aq]=size(add);  add(:,aq)=0;  gInput=gInput+lambda\*add;  else  %nargout = 1  mya= X(i,:)'\*exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  myq =X(i,:)'\*p\*(sech(ip{end}).^2.\*outputWeights(:,m)');  gInput = gInput+(mya-myq)/p;  add=[];  add= inputWeights;  [ap,aq]=size(add);  add(:,aq)=0;  gInput=gInput+lambda\*add;  end  end |

The result on nHidden=[20 20] show as:

Training iteration = 65000, validation error = 0.117200

Training iteration = 70000, validation error = 0.108600

Training iteration = 75000, validation error = 0.110600

Test error with final model = 0.098000

## Task 7:

【Implement “dropout” in which hidden units are dropped out with probability p during training. P=0.5】

### Answer:

|  |  |
| --- | --- |
|  |  |

First i explain the reason of using dropout. Dropout can be realized through randomly selecting some points and enable them not to work.

Why dropout are efficient is that:

1. Everytime the weight is renewed when the hidden point randomly come into appearance. So that we can not ensure every time two neighbouring point appear at the same time. Which means that the renew of weight will not have a steady relationship with the hidden point. It also prevent some features have the particular effect.
2. Dropout has the similar effect as model average. Each sample is input into the network , with the randomly selected hidden point. Each sample has a different model.

When modifying the code ,note that the training way and the testing way should both be adjusted. For the training way the procession of generating of the data and the procession of computing the gradient should both be modified.

|  |
| --- |
| %include dropout  for i = 1:nInstances  dropoutmatrix{1}=(rand(size(X( i ,:)))>0.5);  X(i,:)= X(i,:).\*dropoutmatrix{1};  ip{1} = X(i,:)\*inputWeights;  fp{1} = tanh(ip{1});  for h = 2:length(nHidden)  dropoutmatrix{h}=(rand(size(fp{h-1}))>0.5);  fp{h-1}= fp{h-1}.\*dropoutmatrix{h};  ip{h} = fp{h-1}\*hiddenWeights{h-1};  fp{h} = tanh(ip{h});  end  yhat = fp{end}\*outputWeights;  if length(nHidden) > 1  % Last Layer of Hidden Weights  clear backprop  clear backprop1%/p  backprop = exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights')/p;  backprop1= sech(ip{end}).^2.\*outputWeights(:,m)';  fp{end-1}=fp{end-1}.\*dropoutmatrix{end};  gHidden{end} = gHidden{end} + fp{end-1}'\*backprop-fp{end-1}'\*backprop1+lambda\*hiddenWeights{length(nHidden)-1};  backprop = sum(backprop,1);  % Other Hidden Layers  for h = length(nHidden)-2:-1:1  backprop = (backprop\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  backprop1 = (backprop1\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  fp{h}=ip{h}.\*dropoutmatrix{h+1};  gHidden{h} = gHidden{h} +(fp{h}'\*backprop-fp{h}'\*backprop1)/p+lambda\*(hiddenWeights{h});  end  % Input Weights  backprop = (backprop\*hiddenWeights{1}').\*sech(ip{1}).^2;  backprop1 = (backprop1\*hiddenWeights{1}').\*sech(ip{1}).^2;  X(i,:)=X(i,:).\*dropoutmatrix{1};  gInput = gInput + (X(i,:)'\*backprop- X(i,:)'\*backprop1)/p+ lambda\*(inputWeights);  else  %nargout = 1  X(i,:)=X(i,:).\*dropoutmatrix{1};  mya= X(i,:)'\*exp(yhat)\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  myq =X(i,:)'\*p\*(sech(ip{end}).^2.\*outputWeights(:,m)');  gInput = gInput+(mya-myq)/p+lambda\*(inputWeights);  end |

While for the testing code

|  |
| --- |
| dropoutFraction = 0.5;  % Compute Output  for i = 1:nInstances  dropoutmatrix{1}=(rand(size(X( i ,:)))>0.5);  X(i,:)= X(i,:).\*dropoutmatrix{1};  ip{1} = X(i,:)\*inputWeights;  fp{1} = tanh(ip{1});  for h = 2:length(nHidden)  dropoutmatrix{h}=(rand(size(fp{h-1}))>0.5);  fp{h-1}= fp{h-1}.\*dropoutmatrix{h};  ip{h} = fp{h-1}\*hiddenWeights{h-1};  fp{h} = tanh(ip{h});  end  y(i,:) = fp{end}\*outputWeights; |

And the result shows as :

nHidden=[20]

Training iteration = 65000, validation error = 0.145800

Training iteration = 70000, validation error = 0.132200

Training iteration = 75000, validation error = 0.132400

Test error with final model = 0.119000

From the result we can see athough the validation error is not better than the previous but the training error is better than previous.Dropout does good in generalization ability, so it can prevent overfitting.

## Task 8

【you can do ‘fine-tuning’ of the last layer.Fix the parameters of all the layers except the last one,and solve for the parameters of the last layer exactly as a convex optimization problem.E.g. treat the input to the last layer as the features and use techniques from earlier in the course.(you can use the squared error, since it has a closed-form solution)】

## Answer:

First : I explain why and when we use ‘fine-tuning’.

‘fine-tuning’ is especially useful for training the new data on an existing fine model.

For example: ‘caffe’ use imagenet for training and already get a perfect model. The model classify the pictures into 1000 type. Now I want to apply the model to classify the new picture sample. But this time we just want to classify the pictures into 10 type. How to apply this model on the new sample? We just need to modify the last layer of the network. This is called fine tuning.

Second: I find that there are two factors which affect the result:

1.the size of the new dataset (small or big).

2.its similarity to the original dataset.

|  |  |  |
| --- | --- | --- |
| size of the new dataset (small or big) | similarity to the original dataset. | efficiency |
| small | similiar | No need fine-tuning  but a linear classifier |
| big | similiar | Perfectly prevent overfitting |
| small | different | Linear classifier |
| big | different | Start from the beginning with preparameter. |

Third: So in this particular task. We choice a linear regression model to compute the last layer’s outputweight. the linear regression model already exists in pj1 Here we modify there parts of the code.

|  |
| --- |
| example\_Neural network  for iter = 1:maxIter  ifmod(iter-1,round(maxIter/20)) == 0  [yhat,j] = MLPclassificationPredict2(w,Xvalid,nHidden,nLabels);  k=zeros(size(j,2),nLabels);  yExpanded1=linearInd2Binary(yvalid,nLabels);  for l=1:nLabels  model=leastSquares(j,yExpanded1(:,l));  k(:,l)=model.w;  end  yhat1=j\*k;  [v,yhat1]=max(yhat1,[],2);  fprintf('Training iteration = %d, validation error = %f\n',iter-1, sum(yhat1~=yvalid)/t);  end  % Evaluate test error  [yhat,j]= MLPclassificationPredict2(w,Xtest,nHidden,nLabels);  k=zeros(size(j,2),nLabels);  yExpanded1=LinearInd2Binary(ytest,nLabels);  for l=1:nLabels  model=leastSquares(j,yExpanded1(:,l));  k(:,l)=model.w;  end  yhat1=j\*k;  [v,yhat1]=max(yhat1,[],2);  fprintf('Test error with final model = %f\n',sum(yhat1~=ytest)/t2);  pp=toc;  fprintf('time= %f\n',pp); |

|  |
| --- |
| MLPclassification predict  function [y,j] = MLPclassificationPredict2(w,X,nHidden,nLabels)  [nInstances,nVars] = size(X);  % Compute Output  for i = 1:nInstances  ip{1} = X(i,:)\*inputWeights;  fp{1} = tanh(ip{1});  for h = 2:length(nHidden)  ip{h} = fp{h-1}\*hiddenWeights{h-1};  fp{h} = tanh(ip{h});  end  y(i,:) = fp{end}\*outputWeights;  j(i,:) = fp{end};  end  [v,y] = max(y,[],2);  %y = binary2LinearInd(y); |

|  |
| --- |
| LeastSquares  \_THIS IS FROM PJ.1  function [model] = leastSquares(X,y)  % Solve least squares problem (assumes X'\*X is invertible)  w = (X'\*X)\X'\*y;  model.w = w;  model.predict = @predict;  end |

And the result show as:

Training iteration = 85000, validation error = 0.038800

Training iteration = 90000, validation error = 0.042400

Training iteration = 95000, validation error = 0.039600

Test error with final model = 0.035000 . by changing the last layer ,the result is better than before.

## Task 9:

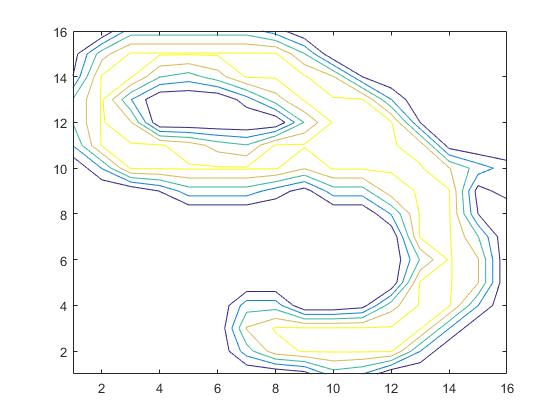
【You can artificially create more training examples,by applying small transformations(translations,rotations,resizing)to the original images】.

### Answer:

In this section:first we see what the **Mnist** data is.

The sample is 5000\*256 ,which means there are 5000 pictures each pictures has a 16\*16 matrix. And the matrix indicates a handwriting-number by using digits to imply the gray value.

|  |
| --- |
| load digits.mat  X(100,:);  Img{1}=reshape(X(100,:),[16,16]);  contour(reshape(X(100,:),[16,16])); |

So we got the picture that:

This is a nine.

And in this task in order to make the trained model more adaptable to various condition. We want to add some noise to the original sample to generate new sample. Combining the original sample with the new sample. We get a more variable sample and expand the Minst.

To realize this thought , what we should do is adding some samples and we can modify the first part of the code.

Also remind that there are several ways to add noise:

**Way 1**

I can add a translation and add some noise.

|  |
| --- |
| ①  X1=X;  for i=1:5000\*256  if X1(i)~=0  X1(i)= X1(i)+10\*(-1+2\*rand);  end  end  ②  for i=1:5000  Img{i}=reshape(X1(i,:),[16,16]);  X1=circshift(X1,1);  X1(1,:)=0;  end  X=[X;X1];  y=[y;y]; |

First part I want to add some noise and the second part is for down translation .

The result shows like this:

Training iteration = 90000, validation error = 0.046200

Training iteration = 95000, validation error = 0.046600

Test error with final model = 0.037000

We can see the validation error going up but the test error declining . which means the model is better to prevent overfitting.

**Way 2**

I want to do the rotation.

|  |
| --- |
| load digits.mat  X=expanddata(X);  y=[y;y];  [n,d] = size(X); |

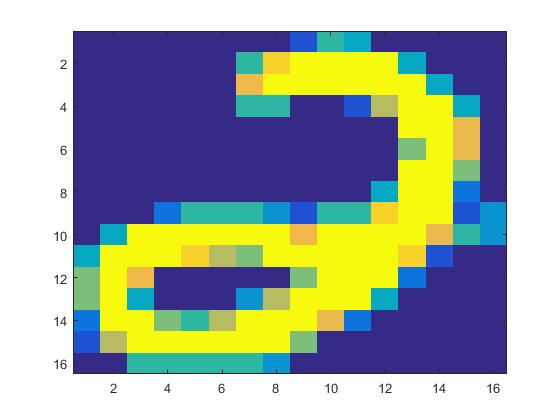
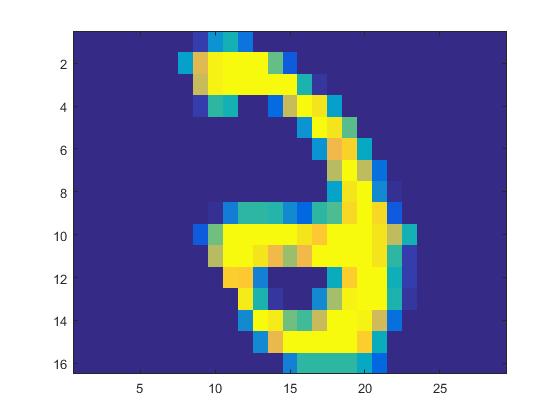
|  |
| --- |
| function [X] = expanddata(X)  for i=1:5000  Img{i}=reshape(X(i,:),[16,16]);  tform=affine2d([0.5 0 0; 0.5 1 0;0 0 1]);%do a translation  J = imwarp(Img{i},tform);  J= reshape(J,[1,256]);  X=[X;J];  end  end |

The affine2d means a 2-D Affine Geometric Transformation. This equals to do the calculation of a 3-D matrix times Xinput



The imwarp means transforming the image A according to the geometric transformation defined by tform.

By modifying the parameter we get when a=c=0.5 or a=0.3 b=0.7 the 16\*16 input matrix transformed into a 16\*16 output matrix.

After carrying out this code: we change the picture like this way and expand the data successfully.

|  |
| --- |
| The code for figure  load digits.mat  X(100,:);  Img{1}=reshape(X(100,:),[16,16]);  figure  imagesc(Img{1})  tform=affine2d([0.5 0 0; .5 1 0; 0 0 1]);  J = imwarp(Img{1},tform);  figure  imagesc(J) |

The result shows like:

Training iteration = 90000, validation error = 0.043000

Training iteration = 95000, validation error = 0.044400

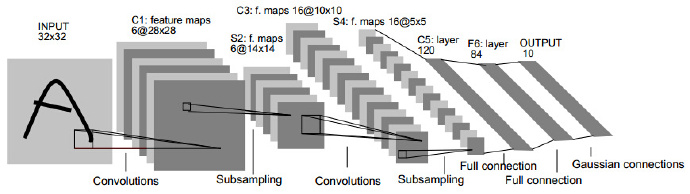
Test error with final model = 0.039000>0.035

It is similar to the first way of expanded data.

## Task 10:

【replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The matlab conv2 function implements 2D convolutions .Filters of size 5 by 5 are a common choice.】

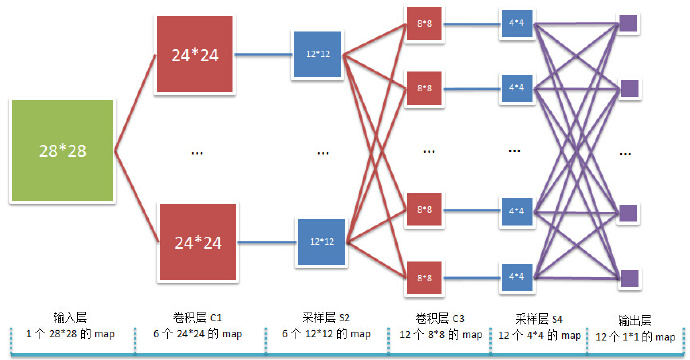
### Answer:



The convlution network can be perfectly explained by this photo.

Forward part:

First we do a matrix multiplication. we times several kernels 5\*5 (for example 8) on the original 5000\*16\*16 matrix. And than we get eight 12\*12 matrix .(in my code I only choose one kernel)

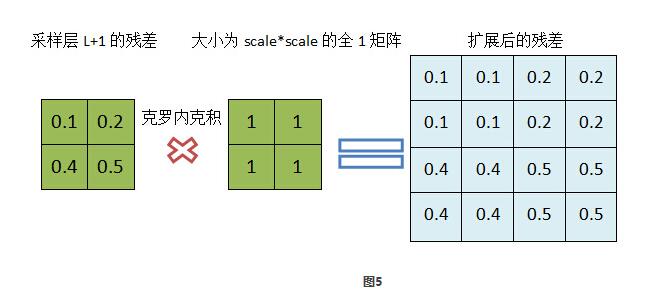


Than we doing a pooling,(in my code I choose an average pooling.) After we use a pooling multiplication we get eight 6\*6 matrix.

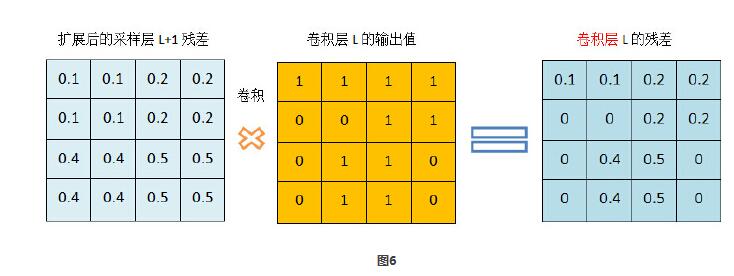
Than we combine these eight 6\*6 matrix and we get a 1\*36\*8 matrix(in my code it is 1\*36). Than iteration again by doing kernel multiplication and pooling multiplication. (in my code I use only one iteration one layer neural network ).

Backprop part:

First the gradient renew in the original network in the order :goutput->ghidden{i}->ginput. Than we have the input backprop which is connected to the CNN network. And than we have the CNN backprop formula.



First using a kron multiplication transfer the 6\*6 pooling output-> 12\*12



Than using an inverse convlution .the code add like this.

convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'valid');

we get the renew of kernel.

My code is divided into three parts:

In the example \_nerual \_network

|  |
| --- |
| % Choose network structure  nHidden = [120 60 30];  ncov=25;%加了25个kernel 的weight有待更新    % Count number of parameters and initialize weights 'w'  nParams = d\*nHidden(1)+25; %加了25的参数！  for h = 2:length(nHidden)  nParams = nParams+nHidden(h-1)\*nHidden(h);  end  nParams = nParams+nHidden(end)\*nLabels;  w = unifrnd(-0.25,0.25,nParams,1); |

In the classifcationLoss 6 function

|  |
| --- |
| function [f,g] = MLPclassificationLoss6(w,X,y,nHidden,nLabels)  lambda=0.02;  [nInstances,nVars] = size(X);  %conv weights  convWeights=reshape(w(1:25),5,5);    %compute conv2  for i=1:nInstances  convInput=reshape(X(i,2:257),16,16);  convOutput=conv2(convInput,convWeights,'valid'); %convOutput是一个12\*12的矩阵,valid使得卷积出来结果对  endend    %pooling(average pooling)  for i=1:nInstances  poolingmatrix=[0.25,0.25;0.25,0.25];  poolingOutput=conv2(convOutput,poolingmatrix,'valid');  poolingOutput=poolingOutput([1,3,5,7,9,11],[1,3,5,7,9,11]);%把poolingOutput变成6\*6的矩阵  endend    %a brandnew 6\*6 input=x把原来的X用卷积pooling以后的6\*6矩阵代替掉，1\*36  X=reshape(poolingOutput,1,36);  [nInstances,nVars] = size(X);  更新ninstances.      % Form Weights  inputWeights = reshape(w(26:25+nVars\*nHidden(1)),nVars,nHidden(1));% %前25个一定是kernel的weight  offset = nVars\*nHidden(1)+25;  for h = 2:length(nHidden)  hiddenWeights{h-1} = reshape(w(offset+1:offset+nHidden(h-1)\*nHidden(h)),nHidden(h-1),nHidden(h));  offset = offset+nHidden(h-1)\*nHidden(h);  end  outputWeights = w(offset+1:offset+nHidden(end)\*nLabels);  outputWeights = reshape(outputWeights,nHidden(end),nLabels);    f = 0;  if nargout > 1  gInput = zeros(size(inputWeights));  for h = 2:length(nHidden)  gHidden{h-1} = zeros(size(hiddenWeights{h-1}));  end  gOutput = zeros(size(outputWeights));  end      % Compute Output    for i = 1:nInstances  ip{1} = X\*inputWeights;  fp{1} = tanh(ip{1});  for h = 2:length(nHidden)  ip{h} = fp{h-1}\*hiddenWeights{h-1};  fp{h} = tanh(ip{h});  end  yhat = fp{end}\*outputWeights;    relativeErr = yhat-y(i,:);  f = f + sum(relativeErr.^2);  if nargout > 1  err = 2\*relativeErr;  % Output Weights  %for c = 1:nLabels  % gOutput(:,c) = gOutput(:,c) + err(c)\*fp{end}';  %end  gOutput = gOutput+(fp{end}'\*err)+lambda\*(outputWeights);  if length(nHidden) > 1  % Last Layer of Hidden Weights  clear backprop  backprop = err\*(repmat(sech(ip{end}).^2,nLabels,1).\*outputWeights');  gHidden{end} = gHidden{end} + fp{end-1}'\*backprop+lambda\*hiddenWeights{length(nHidden)-1};  backprop = sum(backprop,1);  % Other Hidden Layers  for h = length(nHidden)-2:-1:1  backprop = (backprop\*hiddenWeights{h+1}').\*sech(ip{h+1}).^2;  gHidden{h} = gHidden{h} + fp{h}'\*backprop+lambda\*(hiddenWeights{h});  end  % Input Weights  backprop = (backprop\*hiddenWeights{1}').\*sech(ip{1}).^2;  gInput = gInput + X(i,:)'\*backprop+ lambda\*(inputWeights);  **%add a derivative of convweight: 这边是卷积的网络的导数**  **backprop=backprop\*inputWeights'\*1;**  **backprop=reshape(backprop,6,6);**  **expanded\_backprop=kron(backprop,ones(2)); 克罗内克积 gconv=Rot180(conv2(convInput,Rot180(expanded\_backprop),'valid'));**  else  gInput = gInput+X(i,:)'\*err\*(repmat(sech(ip{end}),nLabels,1).^2\*outputWeights')+lambda\*(inputWeights);  这边是卷积的网络的导数  backprop=reshape(backprop,6,6);  这边运用论文的公式，克罗内克积（kron）  expanded\_backprop=kron(backprop,ones(2))；  gconv=Rot180(conv2(reshape(X,6,6),Rot180(expanded\_backprop),'valid'));  end  end  end  % Put Gradient into vector  if nargout > 1  g = zeros(size(w));  1-25位是卷积的权重导数。  g(1:25)=gconv(:);  g(26:nVars\*nHidden(1)+25) = gInput(:);  offset = nVars\*nHidden(1)+25;  for h = 2:length(nHidden)  g(offset+1:offset+nHidden(h-1)\*nHidden(h)) = gHidden{h-1};  offset = offset+nHidden(h-1)\*nHidden(h);  end  g(offset+1:offset+nHidden(end)\*nLabels) = gOutput(:);  end |

For the predict function:

|  |
| --- |
| %改过第十个神经网络的。  function [y] = MLPclassificationPredict3(w,X,nHidden,nLabels)  %先把放入的5000\*256变成6\*6的矩阵。  [nInstances,nVars] = size(X);  %conv weights  convWeights=reshape(w(1:25),5,5);  %compute conv2  for i=1:nInstances%nInstance=1，实际上每一层只传一个参数进来，所以是1  convInput=reshape(X(i,2:257),16,16);  convOutput=conv2(convInput,convWeights,'valid');% convOutput是一个12\*12的矩阵,valid使得卷积出来结果对  end    %pooling(average pooling)%nInstance=1  for i=1:nInstances  poolingmatrix=[0.25,0.25;0.25,0.25];  poolingOutput=conv2(convOutput,poolingmatrix,'valid');  poolingOutput=poolingOutput([1,3,5,7,9,11],[1,3,5,7,9,11]);% 把poolingOutput变成6\*6的矩阵  end    %a brandnew 6\*6 input=x  X=reshape(poolingOutput,1,36);  [nInstances,nVars] = size(X)  %更新instances nvars      % Form Weights  inputWeights = reshape(w(26:nVars\*nHidden(1)+25),nVars,nHidden(1));  offset = nVars\*nHidden(1)+25;  for h = 2:length(nHidden)  hiddenWeights{h-1} = reshape(w(offset+1:offset+nHidden(h-1)\*nHidden(h)),nHidden(h-1),nHidden(h));  offset = offset+nHidden(h-1)\*nHidden(h);  end  outputWeights = w(offset+1:offset+nHidden(end)\*nLabels);  outputWeights = reshape(outputWeights,nHidden(end),nLabels); |

And we need to add a rot180 to make the backprop algorithm reansonable

|  |
| --- |
| function X = Rot180(X)  X = flip(flip(X, 1), 2);  end |

However my written code is a too simply.

But I also found a code using CNN train the Minst. And it is also in our “code document ——CNN”. Besides I read the code and write some annotations in the word.This code is totally an CNN method, in which the

opts.alpha = 1;means the learning rate.

opts.batchsize = 50;means some training data choose for practice

opts.numepochs = 10;means the time running the cnn

Mnist\_Test

epoch 1/10

Elapsed time is 92.843393 seconds.

epoch 2/10

Elapsed time is 88.687072 seconds.

epoch 3/10

Elapsed time is 90.338232 seconds.

epoch 4/10

Elapsed time is 89.018244 seconds.

epoch 5/10

Elapsed time is 90.423627 seconds.

epoch 6/10

Elapsed time is 90.302307 seconds.

epoch 7/10

Elapsed time is 87.644979 seconds.

epoch 8/10

Elapsed time is 89.033128 seconds.

epoch 9/10

Elapsed time is 87.711539 seconds.

epoch 10/10

Elapsed time is 89.928447 seconds.

The result is pretty good which equals 0.0274.

## Task \*

Besides the ten task above there is a particular way to opitimize the result . First we can intialize the weight with in a total different way.

|  |
| --- |
| % Count number of parameters and initialize weights 'w'  nParams = d\*nHidden(1);  for h = 2:length(nHidden)  nParams = nParams+nHidden(h-1)\*nHidden(h);  end  nParams = nParams+nHidden(end)\*nLabels;  w = unifrnd(-0.25,0.25,nParams,1);%use a better way to intialize the w |

And we can have the result greatly improved.

## Conclusion:

In conclusion in this project we have done more than 10 ways to improve the result.

|  |  |
| --- | --- |
| ways to improve score | |
| Ways | **Explanation or realization** |
| Modify the parameters | [nHidden] |
| [weight initialize] |
| [stepsize]=[learning rate ] |
| [lambda] |
| [beta]=[momentum] |
| Change the weight decay expression | Change the momentum strength |
| Change iteration to vector | Rapidly Improve velocity |
| weight penalty&early stop | L2 regularization L1 regularization |
| Change the output unit | Using ‘softmax’ or ‘sigmod’ |
| Adding bias term | Making one hidden unit be constant |
| Using ‘dropout’ | Randomly abandon some points |
| Using ‘fine-tuning’ | ‘linear classifcation’ |
| Creating more training examples | ‘translations’’rotations ’’resizing’ |
| Convolution network | Using kernel as the new weight term. |

According to these methods we have the result as following [in my code].

|  |  |  |
| --- | --- | --- |
| Result report | | |
| Ways' number | Value | Test error |
| 1 | [nHidden]=[120,60,30] | Greatly  Helped  Improving  the score |
| [weight]= unifrnd |
| [stepsize]=if classify |
| [lambda]=0.02 |
| [beta]=if classify |
| 2 | Weight decay | 22% |
| 3 | vectorize | NONE |
| 4 | L2 regularization  Early stop | 13%  11% |
| 5 | sigmod | 8% |
| 6 | bias | 9% |
| 7 | dropout | 11% |
| 8 | Linear classifier | 3.5% |
| 9 | Translation&add noise  rotation | 3.9%  3.7% |
| 10 | Convlution | 2% |

Since my parameter at the first time is not same as the best one which is founded in the latter test. This is not a very reigid comparision .

As you can see from the table , my best result show at the 8 task. It also include the 2,3,4,5 ‘s result and besides the convlution way. [the code has already existed].

The best one is like this.

Training iteration = 0, validation error = 0.371800

Training iteration = 5000, validation error = 0.103800

Training iteration = 10000, validation error = 0.069600

Training iteration = 15000, validation error = 0.059800

Training iteration = 20000, validation error = 0.050400

Training iteration = 25000, validation error = 0.050200

Training iteration = 30000, validation error = 0.046000

Training iteration = 35000, validation error = 0.045400

Training iteration = 40000, validation error = 0.045600

Training iteration = 45000, validation error = 0.044600

Training iteration = 50000, validation error = 0.044000

Training iteration = 55000, validation error = 0.045200

Training iteration = 60000, validation error = 0.040600

Training iteration = 65000, validation error = 0.040600

Training iteration = 70000, validation error = 0.038400

Training iteration = 75000, validation error = 0.040200

Training iteration = 80000, validation error = 0.040800

Training iteration = 85000, validation error = 0.041000

Training iteration = 90000, validation error = 0.038200

Training iteration = 95000, validation error = 0.039800

Test error with final model = 0.035000

time= 281.471113

## Note :

All my code is saved in two places :first the document “题目代码”second the document “做题笔记”which is classified in the order of the 10 tasks.

All my reference word is in “参考”And the pictures are saved in “图片”

## Refenrence

<http://www.open-open.com/lib/view/open1441007579050.html>

<http://cs231n.github.io/optimization-1/>

<https://github.com/rasmusbergpalm/DeepLearnToolbox>