上一个博客已经讲了softmax理论部分，接下来我们就来做个实验，我们有一些手写字体图片（28\*28），训练样本（train-images.idx3-ubyte里面的图像对应train-labels.idx1-ubyte）和测试样本（t10k-images.idx3-ubyte里面的图片对应t10k-labels.idx1-ubyte），我们用训练样本训练softmax模型，测试样本用来做测试。数据和下面讲解的程序下载地址：（[这里](http://pan.baidu.com/s/1mgkO4zQ)）

我们首先展示下我们训练样本部分的图片和label：

1: images = loadMNISTImages('train-images.idx3-ubyte');%得到的images是一个784\*60000的矩阵，意思是每一列是一幅28\*28的图像展成了一列，

2: %一共有60000幅图像。

3: labels = loadMNISTLabels('train-labels.idx1-ubyte');

4: display\_network(images(:,1:100)); % Show the first 100 images

5: disp(labels(1:10));

图片展示：                                                        部分label：4

[](http://images.cnitblog.com/blog/703272/201501/151128417145324.png)                                        [image](http://images.cnitblog.com/blog/703272/201501/151128435425383.png)

我们发现lable是从0~9的，为了便于操作，我们最好转化成1~10（后面转换）。

下面我们进行训练，首先我们定义一些softmax模型常量：

1: inputSize = 28 \* 28; % Size of input vector (MNIST images are 28x28)

2: inputSize =inputSize +1;% softmx的输入还要加上一维（x0=1），也是θj向量的维度

3: numClasses = 10;     % Number of classes (MNIST images fall into 10 classes)

4: lambda = 1e-4; % Weight decay parameter

导入训练样本数据

1: images = loadMNISTImages('train-images.idx3-ubyte');%得到的images是一个784\*60000的矩阵，意思是每一列是一

2: %幅28\*28的图像展成了一列，一共有60000幅图像。

3: labels = loadMNISTLabels('train-labels.idx1-ubyte');

4: labels(labels==0) = 10; % 因为这里类别是1,2..k，从0开始的，所以这里把labels中的0映射成10

5:

6: inputData = images;

7: inputData = [ones(1,60000); inputData];%每个样本都要增加一个x0=1

初始化模型参数：

1: theta = 0.005 \* randn(inputSize\*numClasses, 1);

接下来也是最重要的一步就是：给定模型参数的情况下，求训练样本的softmax的cost function和梯度，即

1: [cost, grad] = softmax\_regression\_vec(theta,inputData ,labels,lambda );

接下来我们就要写softmax\_regression\_vec函数：

1: function [f,g] = softmax\_regression\_vec(theta, X,y,lambda )

2: %下面的n和inputSize指数据有多少维（包括新加的x0=1这一维），也是θj向量的维度

3: %这里y是1,2....到k，从1开始的

4:   m=size(X,2);%X每一列是一个样本，m是指有m个样本

5:   n=size(X,1);  %n指代的前面说了

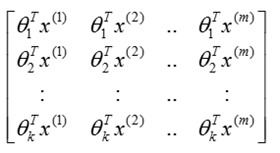
6:   theta=reshape(theta, n, []); %也就是把theta设置成这样矩阵：有inputSize行也就是n行，每一列是一个θj，有k列。这样的θ矩阵跟前面理论部分的θ矩阵不一样，存在  
%转置关系，为什么这样呢？这样这样的话在后面的reshape和矩阵A（:）这样的操作，方便，都是按列进行的，还原也方便。所以只好程序中出现的θ矩阵都是这样的，k列，跟理论部分的相反。

7:   % initialize objective value and gradient.

8:   f = 0;

9:   g = zeros(size(theta));

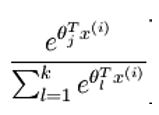
10:   h = theta'\*X;%h是k行m列的矩阵，见图1.

[](http://images.cnitblog.com/blog/703272/201501/151128454487699.png)图1

1: a = exp(h);

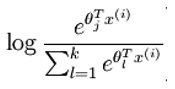
2:  p = bsxfun(@rdivide,a,sum(a)); % sum(a)是一个行向量，每个元素是a矩阵的每一列的和。然后运用bsxfun(@rdivide，，）

3:  %是a矩阵的第i列的每个元素除以 sum(a)向量的第i个元素。得到的p矩阵大小和图1一样，每个元素如图2.

[](http://images.cnitblog.com/blog/703272/201501/151128473086001.png)图2

1: c = log(p); %然后我们取log的对数，c矩阵大小和图1一样，每个元素如图3

要注意取以e为底的对数，如果是其他的最后结果也正确，但是在下面梯度验证 %部分会有一个倍数关系，而不是相等。

[](http://images.cnitblog.com/blog/703272/201501/151128489483302.png)图3

1: i = sub2ind(size(c), y',1:size(c,2));

%y',1:size(c,2)这两个向量必须同时是行向量或列向量

2:   %因为我们接下来每一个样本xi对应的yi是几，就去找到p的每一列中，所对应的第几个元素就是要找的，如图4.首先使用sub2ind

3:   %sub2ind: 在matlab中矩阵是按一列一列的存储的，比如A=[1 2 3;4 5 6]

4: %那么A（2）=4，A(3)=2...而这个函数作用就是比如 sub2ind（size（A）,2,1）就是返回A的第2行第一列的元素存储的下标，因为

5: %A（2）=4，所以存储的下标是2，所以这里返回2.这里sub2ind（size（A）,2,1）的2,1也可以换成向量[a1,a2..],[b1,b2..]但是注意

6: %这两个向量必须同时是行向量或列向量，而不能一个是行向量一个是列向量。所以返回的

7: %第一个元素是A的第a1行第b1列的元素存储的下标，返回的第,二个元素是A的第a2行第b2列的元素存储的下标...i是一个向量，c（i）得到的

8: %向量的每一个元素就是p中每一列你前面要找的的元素。

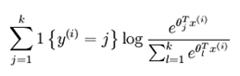
[](http://images.cnitblog.com/blog/703272/201501/151128535114861.png)

图4

1: values = c(i);

2:  f = -(1/m)\*sum(values)+ lambda/2 \* sum(theta(:) .^ 2);  %这个就是cost function

最后求梯度：

1: d = full(sparse(1:m,y,1)); %d为一个稀疏矩阵，有m行k列（k是类别的个数），这个矩阵的（1，y（1））、（2，y（2））

2:   %....(m,y(m))位置都是1。

3:   g = (1/m)\*X\*(p'-d)+ lambda \* theta; %这个g和theta矩阵的结构一样。

4:   g=g(:); % 再还原成向量的形式，这里（：）和reshape都是按列进行的，所以里面位置并没有改变。

解释下这几行：

我们想求梯度矩阵g，这里的g和θ=[θ1,θ2,…,θk]矩阵大小size一样（跟博客中的θ矩阵存在转置关系，之所有代码中这么做，是因为这样再把参数矩阵转成一个向量或转回去利用g（：）或reshape函数按列比较方面），是n行k列的矩阵。n是θj或一个样本xi（包括截距1这一维）的维度大小，k是类别个数。m是样本个数。

我们已经从上一篇博客知道我们g（u，v）的大小通过[image](http://images.cnitblog.com/blog/703272/201501/151128553868934.png)公式求得。

我们知道[image](http://images.cnitblog.com/blog/703272/201501/151128571983222.png)就是前面d矩阵（有m行k列） i 行v列的值，[image](http://images.cnitblog.com/blog/703272/201501/151128584017923.png)是前面p矩阵 （有m列k行）i 列 v行的值。

我们想用矢量编程来求g矩阵：

我们有样本X（代码中每一列是一个样本，也即X为n行m列），那么g = (1/m).\*X\*(p'-d)即是。比如，X的第 i 行乘以（p'-d）的第 j 列就是X（i，j）的值。（正是这种行向量乘以列向量是对应元素相乘再相加就完成了公式里的Σ，这也是矢量编程的核心）

其实cost function不含正则项部分我们还可以用-1/m\*d（：）\*c（：）来实现。

ok,现在我们这个函数写完了，我们想验证下，我们写的这个求导数或着说梯度的这个公式正确不正确，我们还是用之前博客提到的用求导公式来验证，因为你求softmax模型某个参数的导数跟你输入的数据是什么、多少都没有关系，所以我们这有用一些简单的随意写得数据和label，然后随意取一个参数进行验证是不是正确，这些程序在前面已经有了，就不进行讲解了。

1: % DEBUG = true; % Set DEBUG to true when debugging.

2: DEBUG = false;

3: if DEBUG

4:     inputSize = 9;

5:     inputData = randn(8, 100);

7:     inputData = [ones(1,100);inputData];

8:     labels = randi(10, 100, 1);%从[1,100]中随机生成一个100\*1的列向量

10: end

12: % Randomly initialise theta

13: theta = 0.005 \* randn(inputSize\*numClasses, 1);

1: [cost, grad] = softmax\_regression\_vec(theta,inputData ,labels,lambda );

2:

3: if DEBUG

4:  numGrad = computeNumericalGradient( @(theta) softmax\_regression\_vec(theta,inputData ,labels,lambda) ,theta);

5:

6:     % Use this to visually compare the gradients side by side

7:     disp([numGrad grad]);

8:

9:     % Compare numerically computed gradients with those computed analytically

10:     diff = norm(numGrad-grad)/norm(numGrad+grad);

11:     disp(diff);

12:     % The difference should be small.

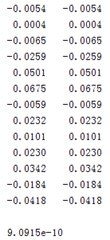
13:     % In our implementation, these values are usually less than 1e-7.

14:

15:     % When your gradients are correct, congratulations!

16: end

然后我们看看检验结果（部分对比）：

[](http://images.cnitblog.com/blog/703272/201501/151128597299895.png)

我们发现最终结果为 9.0915e-10，误差已经足够小了，说明我们的写得函数正确。

接下来，我们要通过迭代算法来最小化我们的cost function，我们仍然使用之前的minfunc 程序包来优化：

1: options.maxIter = 100; %迭代100次

2: softmaxModel = softmaxTrain(inputSize, numClasses, lambda, ...

3:                  inputData, labels, options);

4: %得到的softmaxModel是一个结构体。softmaxModel.optTheta是一个n行k列的参数矩阵，每一列是一个θj

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1: function [softmaxModel] = softmaxTrain(inputSize, numClasses, lambda, inputData, labels, options)

2: % options (optional): options

3: %   options.maxIter: number of iterations to train for

4: if ~exist('options', 'var')

5:     options = struct;

6: end

7: if ~isfield(options, 'maxIter')

8:     options.maxIter = 400;

9: end

10: % initialize parameters

11: theta = 0.005 \* randn(numClasses\* inputSize, 1);

12:

13: % Use minFunc to minimize the function

14: addpath minFunc/

15: options.Method = 'lbfgs'; % Here, we use L-BFGS to optimize our cost

16:           % function. Generally, for minFunc to work, you

17:          % need a function pointer with two outputs: the

18:        % function value and the gradient. In our problem,

19:       % softmaxCost.m satisfies this.

20: minFuncOptions.display = 'on';

21:

22: [softmaxOptTheta, cost] = minFunc( @(theta) softmax\_regression\_vec(theta,inputData,labels,lambda),theta, options);

23:

24: % Fold softmaxOptTheta into a nicer format

25: softmaxModel.optTheta = reshape(softmaxOptTheta,inputSize, numClasses );

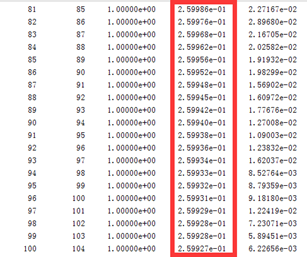
26: softmaxModel.inputSize = inputSize;

27: softmaxModel.numClasses = numClasses;

28:

29: end

然后看下结果：

[](http://images.cnitblog.com/blog/703272/201501/151129040892927.png)

我们发现cost function 的值 收敛在0.2599附近。

接下来来预测我们的预测样本，看看识别的正确率是多少：

1: images = loadMNISTImages('t10k-images.idx3-ubyte');

2: labels = loadMNISTLabels('t10k-labels.idx1-ubyte');

3: labels(labels==0) = 10; % Remap 0 to 10

4:

5: inputData = images;

6: inputData = [ones(1,size(inputData,2)); inputData];%每个样本都要增加一个x0=1

7:

8: [pred] = softmaxPredict(softmaxModel, inputData);

9: acc = mean(labels(:) == pred(:));

10: fprintf('Accuracy: %0.3f%%\n', acc \* 100);

11:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12: function [pred] = softmaxPredict(softmaxModel, data)

13:

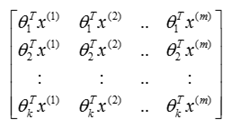
14: theta = softmaxModel.optTheta;  %theta是k列，n行的矩阵

15: pred = zeros(1, size(data, 2));

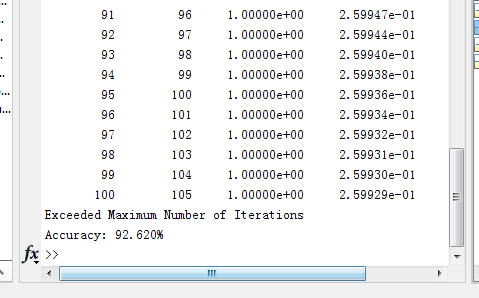
16: [~,pred]= max(theta'\*data);

%theta'\*data这个矩阵如图5，某一个样本softmax最大值与这个矩阵某一列最大

17: %值是等价的，因为每一列除以同一个分母和不除是一样的，并且exp（.）是增函数，所以只求里面的最大值即可。

[](http://images.cnitblog.com/blog/703272/201501/151129086675959.png)图5

最后看一下我们的正确识别率是多少：（我自己运行的结果）



列宁说“世界上除了运动着的**物质**什么都没有”是主张了**世界统一于物质的辩证唯物主义**

这是列宁的一句话，通过以下四点可以帮助你理解这句话：  
1）辩证唯物主义认为，**世界是物质的**，物质世界是运动、变化、发展的。物质是运动的主体，任何运动都是物质的运动，物质是运动的承担者；运动是物质的根本属性和存在方式，它包括一切变化和过程，**任何物质都是运动的物质**。

2）物质和运动是不可分割的：一方面，物质是运动的物质，没有离开运动的物质。这是因为，任何物质都包含内在矛盾，矛盾必然引起运动；物质通过运动而存在。另一方面，运动是物质的运动，没有离开物质的运动。这是因为，物质是一切运动的承担者。物质运动形式是多样的，他们相互区别，并在一定条件下相互转化。

3）设想没有运动的物质，是形而上学的唯物主义世界观；设想没有物质的运动，是唯心主义的世界观。

4）运动是普遍的，永恒的，无条件的，**绝对的**；静止是运动的特殊状态，是**相对的**，有条件的。物质运动在一定条件下，一定范围内可能处于暂时稳定和平衡状态，即相对静止状态，但其运动却没有一刻停止。

**因此，世界上除了运动着的物质之外，什么也没有。**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

题干中的引文是列宁关于世界本原问题的论述，包括两层含义：一是世界是统一的，其本质是物质；二是物质是运动的，运动是物质的本质属性。这些都是辨正唯物主义的基本观点。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

恩格斯指出：“世界的真正统一性在于它的物质性。”

列宁指出：“世界上除了运动着的物质，什么也没有，而运动着的物质只能在空间和时间之内才能运动。”

恩格斯、列宁所说的就是世界的物质统一性原理。

换个角度说，这是名人列宁说的话，我们是不可能认为是错误的，因此不存在否认了什么。

只要知道恩格斯、列宁的原话，无疑就会选择B

19世纪流行于德国的把**唯物主义庸俗化**的思想观点.代表人物有瑞士哲学家、博物学家K.福格特,德国哲学家、医生L.毕希纳,荷兰生理学家J.摩莱肖特.他们承认唯物主义的基本原理,尖锐抨击唯心主义的思辨哲学和宗教.但他们**把意识直接归结为物质**,或认为人的精神活动能力只不过是脑物质的分泌物,或认为思想就是脑髓质的位移.他们在对自然界和人类社会的解释具有强烈的机械论和社会达尔文主义倾向.  
庸俗唯物论是十九世纪三十年代,新黑格尔派解体以后,出现的一个唯物主义哲学派别.它认为宇宙间一切都是物质的,精神也是物质的.这在当时,在反对认为一切都是精神的唯心主义观点上,起过积极的作用.不过,它认为精神这个物质是物质的人脑分泌出来的.说人脑分泌精神就如同肝脏分泌胆汁一样.这就把物质存在的形式庸俗化、简单化、绝对化了.似乎物质的存在只能是实体性的,没有什么特殊的形式.对这一点,连当时的唯物主义者费尔巴哈也不同意,他说如果他们是唯物主义者,我就不做唯物主义者了.我们讲意识是物质的,是讲它是不同于一般物质存在形式的特殊物质形式,即它的物质运动形式.它产生于人脑,从属于人脑,又须臾不能脱离人脑,它不是实体性物质.这都是与庸俗唯物论观点截然不同的.但是,我们还应认识到,庸俗唯物论还应该属于唯物论的范畴,在意识的物质性这个根本点上,它还是正确的.费尔巴哈在批判黑格尔的唯心主义错误时,把其辩证法的核心发展的观点也抽掉了,恩格斯批评费尔巴哈是把婴儿和洗澡水一起泼掉了.所以,我们在批判庸俗唯物论时,也要保护它认为意识是物质的这个正确的根本之点.  
混淆物质与意识的界限,不了解人脑产生意识的过程、抹杀意识本质的一种唯物主义哲学.这种唯物主义产生于19世纪50年代的德国和荷兰,主要代表人物是德国的毕希纳和福格特,荷兰的摩莱肖特.这些人在思维与存在关系问题上,主张存在第一性,思维第二性.但对思维、意识产生过程及其本质,完全作了歪曲地、机械地理解,把意识当作是人脑分泌出来的一种液汁,人脑产生意识如同肝脏分泌胆汁一样.这样,就抹煞了意识与物质的原则区别,把唯物主义庸俗化了.庸俗唯物主义是一种资产阶级哲学思潮,是非科学的世界观.

 庸俗唯物主义承认唯物主义的基本原理，尖锐抨击唯心主义的思辨哲学和宗教。但他们把意识直接归结为物质，或认为人的精神活动能力只不过是脑物质的分泌物，或认为思想就是脑髓质的位移。  
        机械唯物主义即近代形而上学唯物主义，在总结自然科学成就的基础上，丰富和发展了唯物主义。但它把物质归结为自然科学意义上的原子，认为原子是世界的本原，原子的属性就是物质的属性，因而具有机械性，形而上学性，历史观上的唯心主义等局限性。它承认世界的物质性，但却用孤立、静止、片面的观点解释世界，看不到世界上的事物和现象之间的普遍联系与变化发展，或者只是承认机械的联系和机械的运动，因而表现出机械的、形而上学的特征。（世界是由原子和虚无组成的是古希腊伟大唯物主义哲学家德谟克利特的观点，虽然在一定程度上突破了把世界的本原归结为某种或某些具体物质形态的局限性，但仍属于朴素唯物主义的范畴。）