|  |
| --- |
| **Github账号：https://github.com/ZmA0/Coursera\_machine\_learing\_exercise** |
| **实验题目:神经网络模型中算法实现** |
| **实验摘要：**   1. 实验目的：   利用matlab或者octave，掌握对机器学习中的神经网络模型中正向传播，反向传播算法的编程实现。完成对神经网络模型的初步模拟。   1. 实验要求：   利用matlab或者octave的数学编程语言进行书写，按照代码文件中的要求编写并提交文件，使得程序运行通过并且得到足够的分数即为通过。   1. 代码文件要求：   需要编写的文件为  sigmoidGradient.m,  randInitializeWeights.m,  nnCostFunction.m |
| **题目描述**   1. sigmoidGradient.m:   这道题的instructions部分说明我们需要计算S型函数的梯度值，也就是我们常见的导数值，同时还应该满足向量化编程的条件，输入输出数据的格式为矩阵或向量。   1. randInitializeWeights.m   这道题需要我们进行初始化参数的随机化赋值。也就是对神经网络中的权重矩阵即theta进行一个随机化的赋值，同时题目还要求我们要考虑偏倚单元，即矩阵的第一行作为偏倚单元的值。这道题需要利用rand函数进行完成。  3.nnCostFunction.m  这道题需要我们进行神经网络模型的整体初步模拟。题目定义了一个函数要求返回神经网络中的代价函数J以及theta矩阵对应与J的梯度即偏导数。题目要求我们可以按照三个部分完成，第一个进行正向传播的编写求出代价函数J，第二个进行反向传播算法的编写得到theta矩阵的两个偏导数值，第三个进行上述三个值的归一化操作。完成这三步就大致对神经网络算法有了初步的模拟。 |
| **实验过程**   1. sigmoidGradient.m:   根据上面的题目描述可以知道，我们要编写一个函数的偏导数，所以我们可以先利用数学方法进行推演，得到表达式之后，我们直接按照其编写即可。不过，这里需要注意进行乘法运算时，由于向量化编程的条件，因此我们应该用点乘，才能得到正确结果。综上代码入下：  g=sigmoid(z).\*(1-sigmoid(z));   1. randInitializeWeights.m   这道题根据之前所学我们可以直接利用相关公式就可以解决，也就是先生成一个较小的epsilon，然后用这个值去带入公式利用rand函数就可以解决问题。这道题还应该注意为了满足偏倚单元的要求，应该在生成矩阵的列数上加1，因此也就是rand的第二个参数L\_in+1，这样一来，返回值就满足了题意。综上，代码如下：  eps=rand(1);  W=rand(L\_out,L\_in+1)\*2\*eps-eps;   1. nnCostFunction.m   这道题通过上面的描述可以知道整个题目要进行三个部分，这三个步骤可以这么表示  代价函数J  偏导数值  归一化  结束  反向传播算法  正向传播算法  第一部分：编写正向传播算法。通过之前的学习可以知道正向传播求代价函数J主要是一个累加的过程，而神经网络模型中的代价函数是比逻辑回归中的代价函数维度变多了，具体实现过程我们需要先进行模型中激活单元的计算，而这个计算我们需要一层层进行利用公式，然后这样进行计算。同理进行下一层的计算。最后的结果代表第i个样本输出的结果，我们将这个向量进行逻辑回归中的运算，这样一来就完成了一次循环。而代价函数需要我们进行m轮循环的累加才能完成。同时我们还应该注意要在隐藏层和输入层加上 |
| 上偏倚单元a0和X0才能保证正确。  第二部分：反向传播进行偏导数的求解。这部分要求我们进行对上面的代价函数所对应的两个参数矩阵theta1和theta2的两个偏导数。这里我们利用反向传播算法的公式和一层层求出delta的值，最后进行一个累加。这里需要注意的是theta1矩阵需要算到delta2，theta2矩阵需要算到delta3即可。同时还应注意这里计算时应该排除偏倚单元才能得到正确结果。这里的编写也在上面的循环中即可。  第三部分：归一化  这里需要我们队上面的三个值进行归一化处理。代价函数中需要加上theta矩阵的平方和，这里需要先将偏倚单元去除，之后进行点运算求平方，同时也利用matrix(:)这样的操作将矩阵中的值展开方便运算。我们将theta1和theta2这样操作后进行累加即可得到归一化项。偏导数值的归一化同样按照公式加上即可。  综上代码如下：  %this is part1:  X\_new=[ones(m,1) X];%bias unit x0  for i=1:m  a1=X\_new(i,:)';  z2=Theta1 \* a1;  a2=[1;sigmoid(z2)];%bias unit  z3=Theta2 \* a2;  a3=sigmoid(z3);% hidden layer  class=zeros(num\_labels,1);  class(y(i))=1;  J=J+(-1)/m \* sum(class.\*log(a3)+(1-class).\*log(1-a3));    %this is part2  delta3=a3-class;  theta=Theta2(:,2:end)';%except the bias unit  delta2=theta \* delta3.\*sigmoidGradient(z2);  Theta1\_grad=Theta1\_grad+1/m \* delta2.\*a1';  Theta2\_grad=Theta2\_grad+1/m \* delta3.\*a2';  end  %this is part3  Theta1\_new=Theta1;Theta2\_new=Theta2;  Theta1\_new(:,1)=[];  Theta2\_new(:,1)=[];  s1=Theta1\_new.^2;  s2=Theta2\_new.^2;  S=lambda/(2\*m) \* (sum(s1(:)) + sum(s2(:)));  J=J+S;  Theta1(:,1)=0; Theta2(:,1)=0;  Theta1\_grad=Theta1\_grad+lambda/m \* Theta1;  Theta2\_grad=Theta2\_grad+lambda/m \* Theta2; |
| S=lambda/(2\*m) \* (sum(s1(:)) + sum(s2(:)));  J=J+S;  Theta1(:,1)=0; Theta2(:,1)=0;  Theta1\_grad=Theta1\_grad+lambda/m \* Theta1;  Theta2\_grad=Theta2\_grad+lambda/m \* Theta2;  **实验总结**  第四次编程中第三题对神经网络中代价函数的编写进行了整体的编写，难度比较大，而且写的时候也出现了许多错误。总体而言虽然题目给的提示比较多，但是涉及的公式较多，参数也比较多，所以编写的时候很不容易将其的关系调整清楚，而且需要进行向量化的编写。这次编写和整个题目文件多结合了一点，边编写边看着题目文件所给的数据，即ex4data和ex4weights这两个，包含了必要的输入输出参数和矩阵参数，有一个直观的维度的展示，而且这样更能够将代码和实际数据结合起来。 |
| **参考文献**  **<https://wenku.baidu.com/view/f328b62b69dc5022abea0068.html>**  （同样参考了这个笔记）  <http://blog.csdn.net/firethelife/article/details/51326931>  （反向传播算法推演）  <https://www.zhihu.com/question/24827633>  <https://www.zhihu.com/question/27239198?rf=24827633>  （反向传播算法相关知乎问题） |