编程练习4：神经网络学习

介绍

在本练习中，你将实现神经网络的反向传播算法，并将其应用于手写数字识别任务。

要开始练习，使用Octave中的cd命令更改到此目录。

此练习中包含的文件

ex4.m - Octave脚本，帮助你逐步完成练习

ex4data1.mat - 手写数字训练集

ex4weights.mat- 练习4的神经网络参数

submitWeb.m -可选提交脚本

displayData.m-函数帮助可视化数据集

fmincg.m-函数最小化例程（类似于fminunc）

sigmoid.m - sigmoid函数

computeNumericalGradient.m - 数值计算梯度

checkNNGradients.m - 帮助检查梯度的函数

debugInitializeWeights.m - 初始化权重的函数

predict.m-神经网络预测函数

[\*]sigmoidGradient.m - 计算sigmoid函数的梯度?

[\*]randInitializeWeights.m - 随机初始化权重?

[\*]nnCostFunction.m - 神经网络代价函数?

\* 指示需要完成的文件

在整个练习中，你将使用脚本ex4.m。这些脚本为问题设置数据集，并调用你编写的函数。不需要修改脚本。你只需按照本作业中的说明修改其他文件中的函数。

1 神经网络

在上一个练习中，你实现了神经网络的前馈传播，并使用它使用我们提供的权重预测手写数字。在本练习中，你将实现反向传播算法来学习神经网络的参数。

提供的脚本ex4.m将帮助你逐步完成此练习。

1.1 可视化数据

在ex4.m的第一部分中，代码将通过调用函数displayData加载数据并将其显示在二维图（图1）上。

这与上一个练习中使用的数据集相同。ex3data1.mat中有5000个训练示例，其中每个训练示例是数字的20像素乘20像素灰度图像。每个像素由

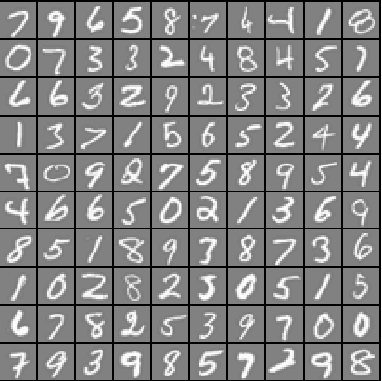
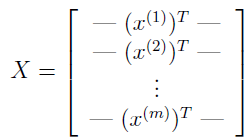


图1：数据集示例

表示该位置灰度强度的浮点数。像素的20乘20网格被“展开”成400维向量。这些训练示例中的每一个都成为我们的数据矩阵X中的一行。这给了我们一个5000×400的矩阵X，其中每一行都是手写数字图像的训练示例。

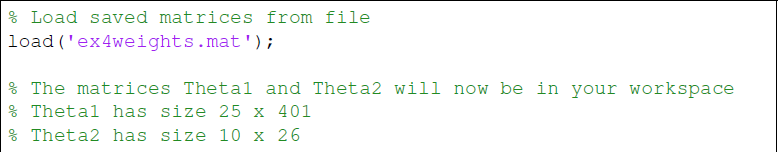


训练集的第二部分是包含训练集标签的5000维向量y。为了使事情与八度/Matlab索引更加兼容，在没有零索引的情况下，我们将数字0映射到值10。因此，“0”数字标记为“10”，而数字“1”到“9”按其自然顺序标记为“1”到“9”。

1.2 模型表示法

我们的神经网络如图2所示。它有3层-一个输入层，一个隐藏层和一个输出层。回想一下，我们的输入是数字图像的像素值。由于图像的大小是20×20，这给了我们400个输入层单元（不包括总是输出+1的额外偏置单元）。训练数据将通过ex4.m脚本加载到变量X和y中。

我们已经向你提供了一组网络参数（Θ（1）、Θ（2）），这些存储在ex4weights.mat中的参数，将由ex4.m加载到θ1和θ2中。这些参数的维数是为第二层有25个单元和10个输出单元（对应于10位类）的神经网络确定的。



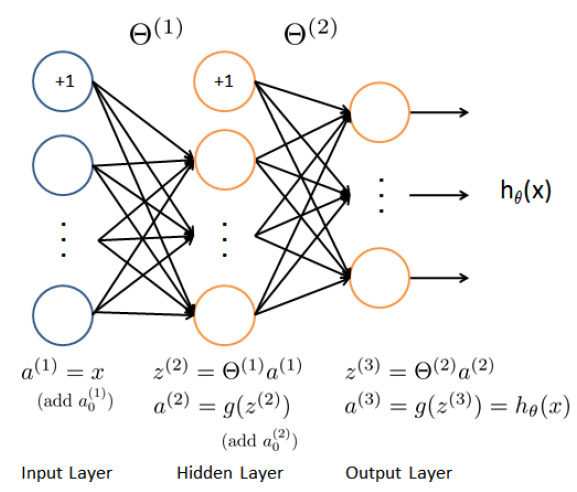
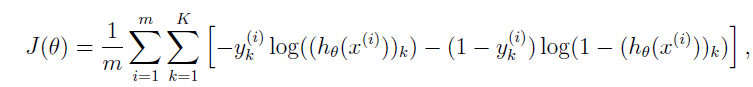


图2：神经网络模型。

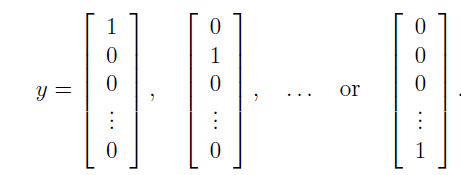
1.3 前馈与成本函数

现在你将实现神经网络的代价函数和梯度。首先，完成nnCostFunction.m中的代码以返回成本。

回想一下，神经网络的代价函数（没有正则化）是

 ,

其中hθ（x（i））如图2所示计算，K=10是可能的标签总数。注意，hθ（x（i））k＝a（3）k是第k输出单元的激活（输出值）。另外，回想一下，虽然原始标签（在变量y中）是1，2，…，10，但是为了训练神经网络，我们需要将标签重新编码为只包含值0或1的向量，以便

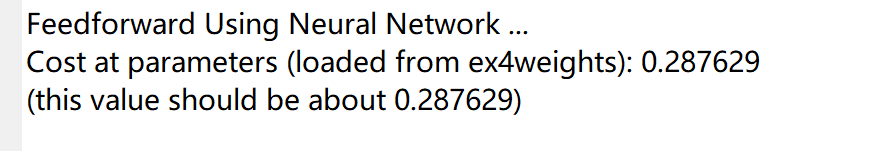


例如，如果x（i）是数字5的图像，那么对应的y（i）（应该与代价函数一起使用）应该是y5=1的10维向量，其他元素等于0。

应该实现前馈计算，为每个示例i计算hθ（x（i）），并对所有示例求和成本。你的代码还应该适用于具有任意数量标签的任意大小的数据集（你可以假设至少有K≥3个标签）。

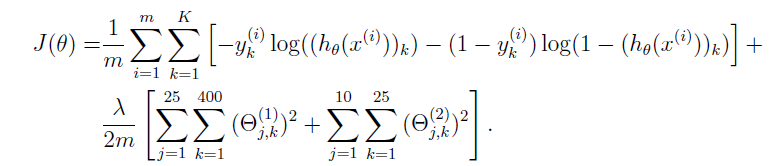
实施说明：矩阵X包含行中的示例（即，X（i，：）&apos;是第i个训练示例X（i），表示为n×1向量）。在nnCostFunction.m中完成代码时，需要将1的列添加到X矩阵中。神经网络中每个单元的参数用θ1和θ2表示为一行。具体地，θ1的第一行对应于第二层中的第一隐藏单元。您可以在示例上使用for循环来计算成本。

完成后，ex4.m将使用加载的θ1和θ2参数集调用nncost函数。你应该看到成本大约是0.287629。



1.4 正则化代价函数

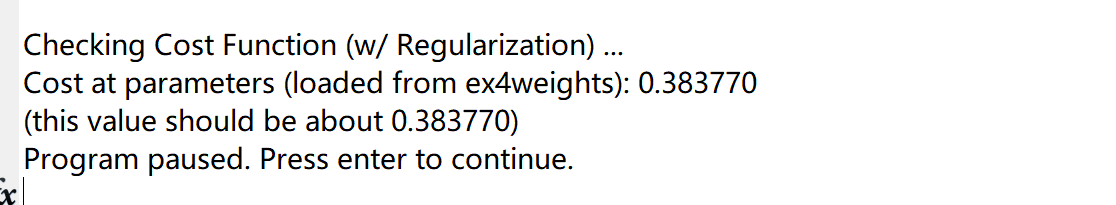
正则化神经网络的代价函数由下式给出



你可以假设神经网络只有3层——输入层、隐藏层和输出层。但是，你的代码应该适用于任意数量的输入单元、隐藏单元和输出单元。虽然为了清晰起见，我们已经明确列出了上面关于Θ（1）和Θ（2）的索引，但请注意，您的代码通常应该使用任何大小的Θ（1）和Θ（2）。

请注意，你不应规范化与偏差相对应的术语。对于矩阵θ1和θ2，这对应于每个矩阵的第一列。现在应该将正则化添加到代价函数中。请注意，可以首先使用现有的nnCostFunction.m计算未正则化的代价函数J，然后为正则化项添加代价。

完成后，ex4.m将使用加载的θ1和θ2参数集调用nncost函数，λ=1。你应该看到成本大约是0.383770。



2 反向传播

在这部分练习中，你将实现反向传播算法来计算神经网络代价函数的梯度。你需要完成nnCostFunction.m，以便它为grad返回适当的值。一旦你计算了梯度，你就可以通过使用诸如fmincg之类的高级优化器最小化代价函数J（Θ）来训练神经网络。

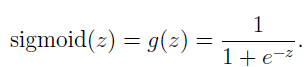
你将首先实现反向传播算法来计算（未规范化）神经网络参数的梯度。在验证了非正则化情况下的梯度计算是正确的之后，你将实现正则化神经网络的梯度。

2.1 sigmoid梯度

为了帮助您开始这部分练习，你将首先实现sigmoid梯度函数。sigmoid函数的梯度可以计算



其中



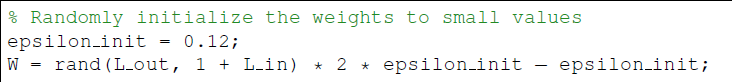
。

完成后，尝试在Octave命令行调用sigmoidGradient（z）来测试一些值。对于z的值（正数和负数），梯度应接近0。当z=0时，梯度应正好为0.25。你的代码也应该使用向量和矩阵。对于矩阵，函数应该对每个元素执行sigmoid梯度函数。

2.2 随机初始化

在训练神经网络时，随机初始化对称性破缺的参数是非常重要的。随机初始化的一个有效策略是随机选择范围内的Θ（l）的值。您应该使用。此范围的值，以确保参数保持较小，并使学习更有效。

你的工作是完成randInitializeWeights.m以初始化Θ的权重；修改文件并填写以下代码：



2.3 反向传播

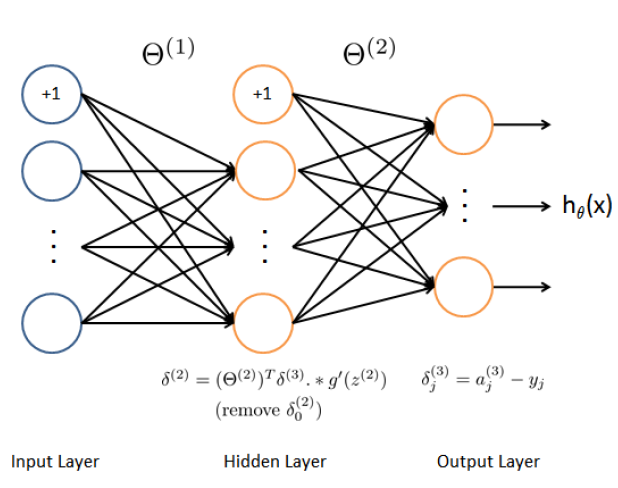


图3：反向传播更新。

现在，你将实现反向传播算法。回想一下，反向传播算法背后的直觉如下。给定一个训练示例（x（）t，y（）t），我们将首先运行“向前传递”来计算整个网络中的所有激活，包括假设hΘ（x）的输出值。然后，对于层l中的每个节点j，我们要计算一个“错误项”，它度量该节点对输出中的任何错误“负责”的程度。

对于输出节点，我们可以直接测量网络的激活和真实目标值之间的差异，并用它来定义（因为第3层是输出层）。对于隐藏单元，你将根据层（l+1）中节点的误差项的加权平均值进行计算。

具体来说，这里是反向传播算法（也在图3中描述）。你应该在一次处理一个示例的循环中实现步骤1到4。具体来说，你应该为t=1:m实现for循环，并将下面的步骤1-4放在for循环内，迭代在训练示例（x（t），y（t））上执行计算。第5步将累计梯度除以m，得到神经网络代价函数的梯度。

1. 将输入层的值（a（1））设置为第t个训练示例x（t）。

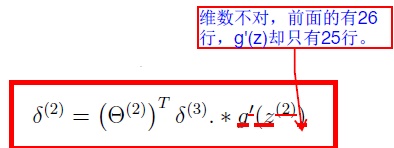
执行前馈过程（图2），计算第2层和第3层的激活。请注意，你需要添加+1项，以确保层a（1）和a（2）的激活向量也包括偏置单元。在八度音阶中，如果1是一个列向量，加一对应于1=[1；a1]。

2. 对于第3层（输出层）中的每个输出单元k，设置



其中yk∈{0,1}表示当前训练示例是属于类k（yk=1），还是属于不同的类（yk=0）。你可能会发现逻辑数组对此任务很有帮助（在前面的编程练习中解释）。

3. 对于隐藏层l=2，设置



4. 使用以下公式累积本例中的梯度。请注意，你应该跳过或删除。在八度音阶中，删除对应于delta 2=增量 2（2:结束）。



5. 通过将累积梯度除以得到神经网络的代价函数梯度：



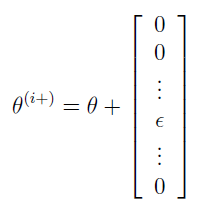
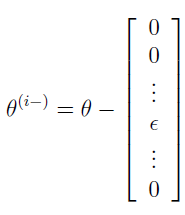
Octave 建议：只有在成功完成前馈函数和代价函数之后，才能实现反向传播算法。在实现反向传播算法时，如果遇到维度不匹配错误（“非一致参数”Octave错误），使用size函数打印出正在处理的变量的大小通常很有用。

在实现了反向传播算法之后，脚本ex4.m将继续对实现运行梯度检查。梯度检查将允许你增加代码正确计算梯度。

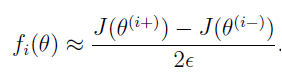
2.4 梯度检查

在你的神经网络中，你是最小化代价函数J（Θ）。要对参数执行梯度检查，可以想象将参数Θ（1）、Θ（2）展开为长向量θ。通过这样做，你可以认为代价函数是J（θ），而不是使用下面的梯度检查过程。

假设你有一个函数fi（θ），据说是用来计算的；你想检查fi是否输出了正确的导数值。

设和

所以，θ（i+）与θ相同，只是它的第i个元素增加了。类似地，θ（i−）是第i个元素减少的对应向量。现在可以通过检查每个i的值来验证fi（θ）的正确性：

 .

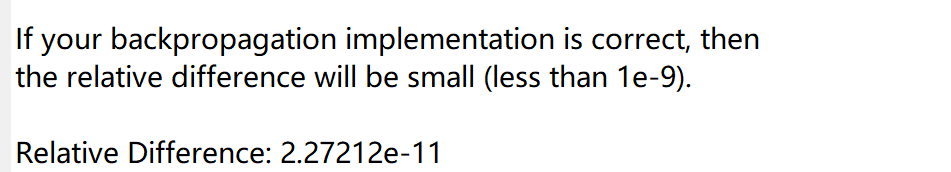
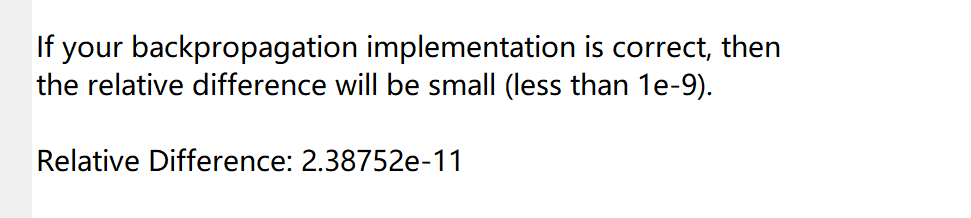
这两个值彼此接近的程度将取决于J的细节。但是假设，你通常会发现上面的左右两侧至少有4个有效数字（通常更多）。

我们已经在computeNumericalGradient.m中为你实现了计算数值梯度的函数。虽然你不需要修改该文件，但我们强烈建议你查看代码以了解其工作原理。

在ex4.m的下一步中，它将运行提供的函数checkNNGradients.m，该函数将创建一个小的神经网络和数据集，用于检查梯度。如果你的反向传播实现是正确的，你应该看到一个小于1e-9的相对差异。

实用提示：在执行梯度检查时，使用具有相对较少的输入单元和隐藏单元的小型神经网络更有效，因此具有相对较少的参数。θ的每个维度需要对代价函数进行两次评估，这可能会很昂贵。在函数checkNNGradients中，我们的代码创建了一个小的随机模型和数据集，与computeNumericalGradient一起用于梯度检查。此外，在你确信梯度计算是正确的之后，你应该在运行学习算法之前关闭梯度检查。

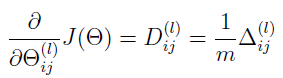
实用提示：梯度检查适用于计算成本和梯度的任何函数。具体来说，你可以使用相同的computeNumericalGradient.m函数来检查其他练习的梯度实现是否也正确（例如，logistic回归的成本函数）。

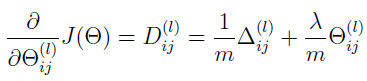


2.5 正则化神经网络

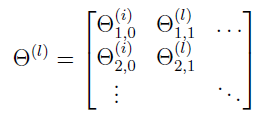
成功实现反向传播算法后，将向梯度添加正则化。为了说明正则化，可以在使用反向传播计算梯度之后将其作为附加项添加。

具体来说，在使用反向传播计算∆（ijl）之后，应该使用

 对于j=0

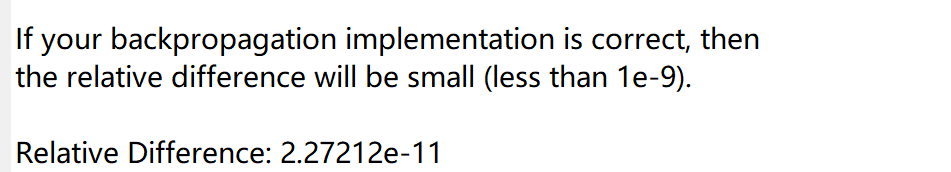
 对于j≥1

请注意，不应正则化用于偏差项的Θ（）l的第一列。此外，在参数中，Θ从1开始索引，j从0开始索引。因此，

 .

有些令人困惑的是，倍频程中的索引从1开始（对于i和j），因此θ1（2，1）实际上对应于Θ（2l，）0（即，上面所示矩阵Θ（1）的第二行第一列中的条目）

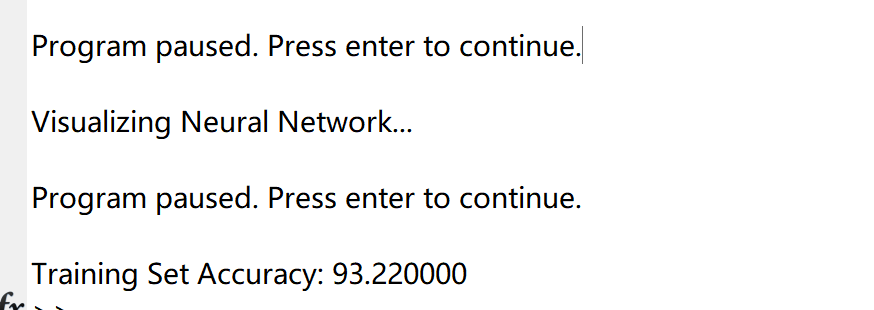
现在修改计算nnCostFunction中grad的代码以考虑正则化。完成后，ex4.m脚本将继续对实现运行渐变检查。如果你的代码是正确的， 应该期望看到小于1e-9的相对差异。



2.6 使用fmincg学习参数

在成功实现了神经网络代价函数和梯度计算之后，ex4.m脚本的下一步将使用fmincg来学习一组好的参数。

训练完成后，ex4.m脚本将继续通过计算正确示例的百分比来报告分类器的训练精度。如果你的实现是正确的，你将看到报告的训练精度约为95.3%（由于随机初始化，这可能会变化约1%）。通过对神经网络进行多次迭代训练，可以获得较高的训练精度。我们鼓励你尝试训练神经网络进行更多的迭代（例如，将MaxIter设置为400），并改变正则化参数λ。通过正确的学习设置，可以使神经网络完全适合训练集。



3 可视化隐藏层

理解你的神经网络在学习什么的一种方法是将隐藏单元捕捉到的表示可视化。非正式地说，给定一个特定的隐藏单元，可视化它所计算内容的一种方法是找到一个将使它激活的输入x（即，有一个激活值接近1）。对于你训练的神经网络，请注意，Θ（1）的第i行是一个401维向量，它表示第i个隐藏单元的参数。如果我们去掉偏置项，我们得到一个400维向量，它表示从每个输入像素到隐藏单元的权重。

因此，可视化由隐藏单元捕获的“表示”的一种方法是将该400维向量重塑为20×20图像并显示是的。是的ex4.m的下一步是使用displayData函数来实现这一点，它将向你显示一个包含25个单元的图像（类似于图4），每个单元对应于网络中的一个隐藏单元。[1]

在经过训练的网络中，你应该发现隐藏的单位大致对应于在输入中查找笔划和其他模式的检测器。

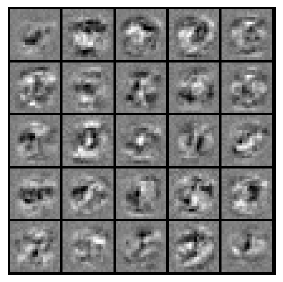


图4：隐藏单元的可视化。

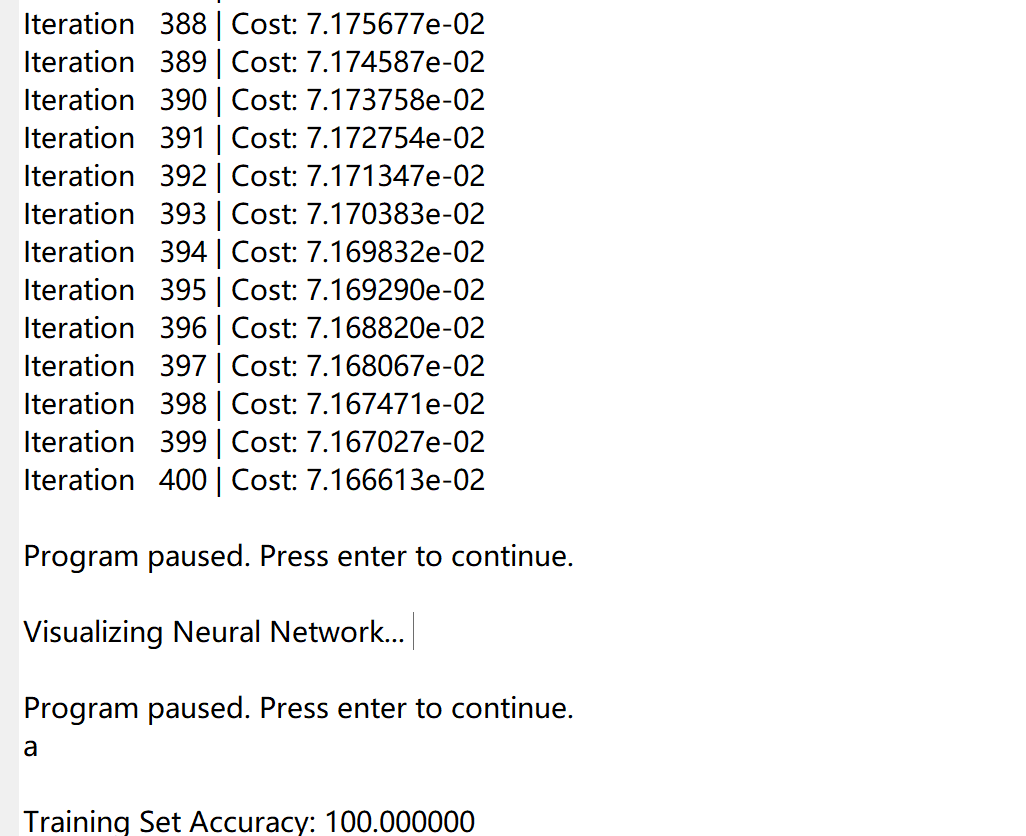
3.1 可选（未分级）练习

在这部分练习中，你将尝试神经网络的不同学习设置，以了解神经网络的性能如何随正则化参数λ和训练步骤数（参数）而变化

使用fmincg时的MaxIter选项）。

神经网络是非常强大的模型，可以形成高度复杂的决策边界。在没有正则化的情况下，神经网络有可能“过度拟合”训练集，从而在训练集上获得接近100%的准确率，但在新的例子上却没有以前见过的那样好。你可以将正则化λ设置为较小的值，将MaxIter参数设置为较高的迭代次数，以便自己看到这一点。

当你改变学习参数λ和MaxIter时，你也能亲眼看到隐藏单位的可视化变化。



当lambda = 0.1, maxlter = 400 时,得到准确率为100%,但是确实“过度拟合”。