编程练习 4：神经网络学习

机器学习

# 介绍

在本练习中，您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的任务。在开始编程练习之前，我们强烈建议您观看视频讲座并完成 相关主题的评论问题。

要开始练习，您需要下载入门代码并将其内容解压缩到要完成练习的目录中。如果需要，请在开始本练习之前使用 Octave/MATLAB 中的 cd 命令更改为此目录。

您还可以在课程网站的“环境设置说明”中找到有关安装Octave / MATLAB的说明。

## 本练习中包含的文件

ex4.m - Octave/MATLAB 脚本，可引导您完成练习 ex4data1.mat - 训练一组手写数字 ex4weights.mat - 练习 4 submit.m 的神经网络参数 - 将您的解决方案发送到我们服务器的提交脚本 displayData.m - 帮助可视化数据集的功能 fmincg.m - 函数最小化例程（类似于 fminunc）sigmoid.m - Sigmoid 函数 computeNumericalGradient.m - 数值计算梯度检查NNGradients.m - 帮助检查梯度 debug的函数初始化权重.m - 函数用于初始zing 权重 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 - 神经网络成本函数

*?* 表示您需要完成的文件

在整个练习过程中，您将使用 脚本 ex4.m。 这些脚本为问题设置数据集，并调用您将编写的函数。您不需要修改脚本。您只需按照此分配中的说明修改其他文件中的函数。

## 从何处获取帮助

本课程中的练习使用Octave[[1]](#footnote-1) 或MATLAB，这是一种非常适合数值计算的高级编程语言。如果您没有 安装Octave或MATLAB，请参阅课程网站“环境设置说明”中的安装说明。

在 Octave/MATLAB 命令行中，键入 help 后跟函数名称，将显示内置 function 的文档。例如，帮助绘图将显示绘图的帮助信息。有关倍频程函数的更多文档，请参阅[倍频程文档页面。](http://www.gnu.org/software/octave/doc/interpreter/) MATLAB 文档可以在 [MATLAB 文档页面上](http://www.mathworks.com/help/matlab/?refresh=true)找到[。](http://www.mathworks.com/help/matlab/?refresh=true)

我们还强烈建议使用在线 **讨论** 与其他学生讨论练习。但是，不要查看他人编写的任何源代码或与他人共享您的源代码。

# 神经网络

在前面的练习中，您为神经网络实现了前馈传播，并使用它来预测具有我们提供的权重的手写数字。在本练习中，您将实现反向传播离子算法来 *学习* 神经网络的参数。

提供的脚本 ex4.m 将帮助您逐步完成本练习。

## 可视化数据

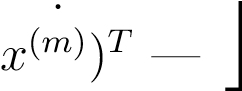
在 ex4.m 的第一部分中，代码将通过调用函数 displayData 加载数据并将其显示在二维图上（图 1）。



图 1：数据集中的示例

这与您在上一练习中使用的数据集相同。ex3data1.mat 中有 5000 个训练示例，其中每个训练示例都是数字的 20 x 20 像素灰度图像。每个像素由一个浮点数表示，该浮点数指示该位置的灰度强度。20 x 20 像素网格被“展开”成一个 400 维矢量。这些训练示例中的每一个都成为数据矩阵 X 中的一行。这给了我们一个5000×400矩阵X，其中每一行都是手写数字图像的训练示例。

 — （*x*（1））*T* — 

— （*x*（2）*）T* —

 ..



— (

训练集的第二部分是一个 5000 维向量 y ，其中包含训练集的标签。为了使内容与Octave / MATLAB索引更加兼容，其中没有零索引，我们将数字零映射到值十。因此，e，“0”数字被标记为“10”，而数字“1”到“9”按自然顺序标记为“1”到“9”。

## 模型表示

我们的神经网络如图 2 所示。它有3层 - 输入层，隐藏层和输出层。回想一下，我们的输入是数字图像的像素值。由于图像的大小为20×20，因此为我们提供了400个输入层单位（不包括始终输出+1的额外偏置单位）。训练数据将由 ex4.m script 加载到变量 X 和 y 中。

您已经为您提供了一组已由我们训练的网络参数（Θ（1）*，*Θ（2））。这些存储在ex4weights.mat中，并由ex4.m加载到Theta1和Theta2中。这些参数的维度适用于神经网络，第二层有 2 个 5 个单位，10 个输出单位（对应于 10 位数字类）。

|  |
| --- |
| % 从文件加载中加载保存的矩阵 （'ex4weights.mat'）;  % 矩阵 Theta1 和 Theta2 现在将位于工作区中  % Theta1 的尺寸为 25 x 401  % Theta2 的尺寸为 10 x 26 |

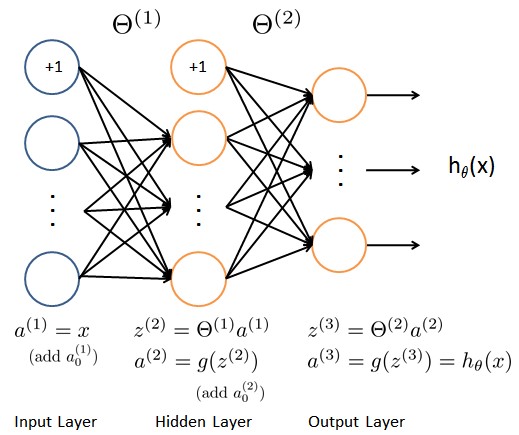
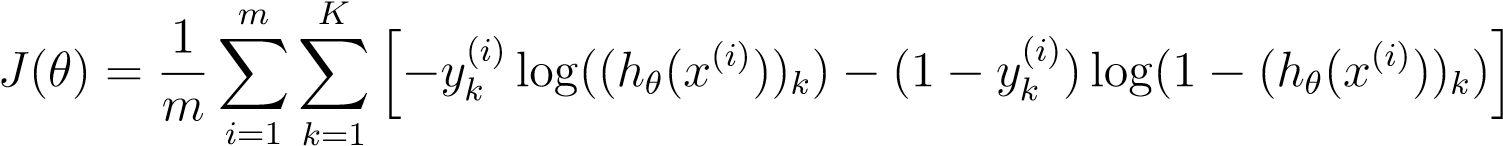


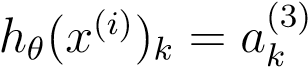
图 2：神经网络模型。

## 前馈和成本函数

现在，您将实现神经网络的成本函数和梯度。首先，完成 nnCostFunction.m 中的代码以返回成本。

回想一下，神经网络的成本函数（不正则化）是

 *,*

其中 *hθ*（*x*（*i*）） 计算为图 2 所示的 s，*K* = 10 是可能的标签总数。请注意，这是第 *k* 个输出单元的激活（输出值）。 另外，回想一下，虽然原始标签（在变量y中）是1，2，...，10，为了训练神经网络g，我们需要将标签重新编码为仅包含值0或1的向量，以便

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|    1   0      *y* =  0 *，*   ...       0 |    0   1       0 *，*   ...       0 | *...* | 或 |    0   0      0 *.*   ...       1 |

例如，如果 *x*（*i*） 是数字 5 的图像，则相应的 *y*（*i*）（应与成本函数一起使用）应为 y 5 = 1 的 10 维向量 ，其他元素等于 0。

您应该实现前馈计算，为每个示例 *i* 计算 *hθ*（*x*（*i*）），并对所有示例的成本求和。 **您的代码还应该适用于任何大小、具有任意数量标签的数据集**（您可以假设始终至少有 *K* 个≥ 3 个标签）。

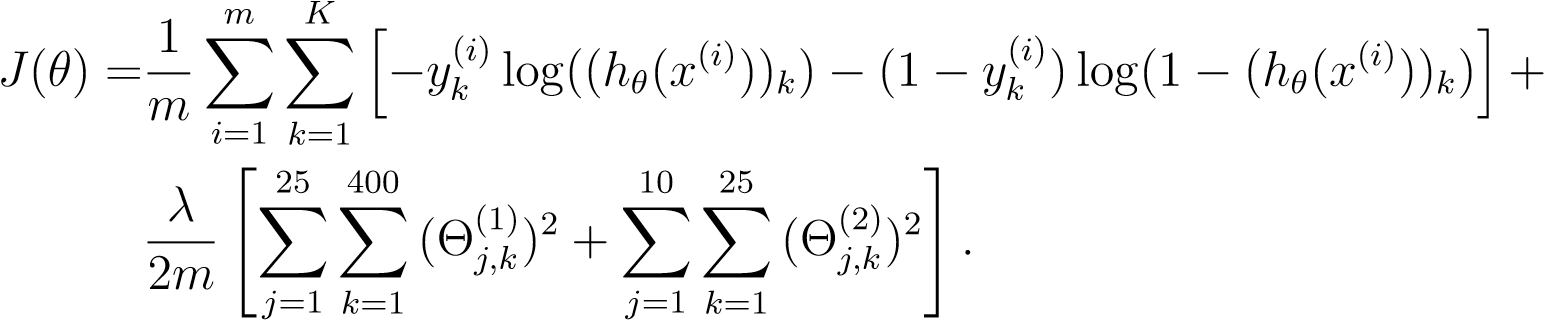
**实现说明：**矩阵 X 包含行中的示例（即 X（i，:)' 是第 i 个训练示例 *x*（*i*），表示为 *n* × 1 向量。当您在 nnCostFunction.m 中完成代码时，您需要将 1 列添加到 X matrix 中。神经网络中每个单元的参数在 Theta1 和 Theta2 中表示为一行。具体来说，Theta1 的第一行对应于第二层中的第一个隐藏单元。您可以在示例上使用 for 循环来计算成本。

完成后，ex4.m 将使用为 Theta1 和 Theta2 加载的参数集调用您的 nnCostFunction。您应该看到成本约为 0.287629。

*您现在应该提交您的解决方案。*

## 正则化成本函数

具有正则化的神经网络的成本函数由下式给出



您可以假设神经网络只有3层 - 输入层，隐藏层和输出层。但是，您的代码应适用于任意数量的输入单元、隐藏n 个单元和输出单元。虽然我们为了清楚起见，已经明确列出了上面 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，不应正则化与偏差相对应的项。对于矩阵 Theta1 和 Theta2，这对应于每个矩阵的第一列。现在，应将正则化添加到成本函数。请注意，您可以先使用现有的 nnCostFunction.m 计算未计量的成本函数 *J*  ，然后再添加正则化项的成本。

完成后，ex4.m 将使用加载的 Theta1 和 Theta2 参数集调用您的 nnCostFunction，并且 *λ* = 1。您应该看到成本约为 0.383770。

*您现在应该提交 您的解决方案。*

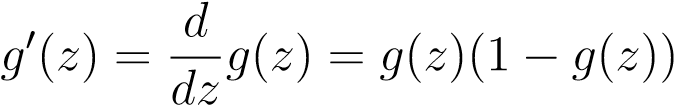
# 反向传播

在本部分练习中，您将实现反向传播算法来计算神经网络成本函数的梯度。您需要完成 nnCostFunction.m，以便为 grad 返回适当的 value。计算梯度后，您将能够通过使用高级优化器（如 fmincg）最小化成本函数 *J*（Θ） 来训练神经网络。

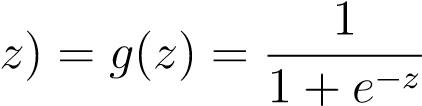
您将首先实现反向传播算法，以计算（未映射）神经网络的参数的梯度 s。在验证未实例的梯度计算正确后，您将为正则化神经网络实现梯度。

## 乙状结肠梯度

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



哪里

sigmoid（*.*

完成后，尝试通过在Octave/MATLAB命令行调用sigmoidGradient（z）来测试几个值。对于 z 的大值（正值和负值），梯度应接近 0。当 z = 0 时，梯度应正好为 0.25。您的代码还应该使用向量和矩阵。对于矩阵，您的函数应在每个元素上执行 Sigmoid 梯度函数。

*您现在应该提交您的解决方案。*

## 随机初始化

在训练g神经网络时，随机初始化对称性破坏的参数非常重要。随机初始化的一种有效策略是在[

您应该使用此 值范围来确保参数处的 th 保持较小，并使学习更加高效。

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

|  |
| --- |
| % 随机将权重初始化为小值 epsiloninit = 0.12;  W = rand（L out， 1 + L in） \* [[2]](#footnote-2) \* epsiloninit − epsiloninit; |

*您无需为本部分练习提交任何代码。*

## 反向传播

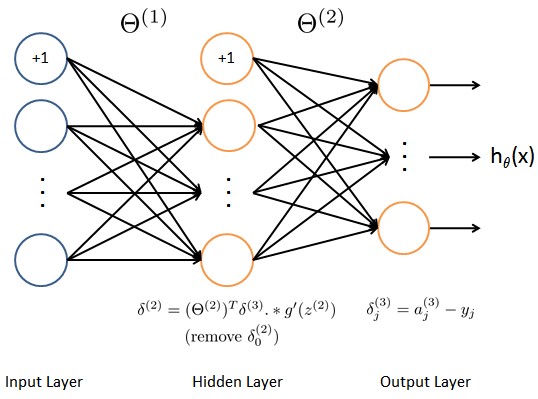


图 3：反向传播更新。

现在，您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*）），我们将首先运行一个“正向传递”来计算整个网络中的所有激活，包括假设 *h*Θ（*x*） 的输出值。然后，对于l层中的每个节点*j*，我们想要计算一个“误差项”，该项  测量该节点w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 *δj*（3）

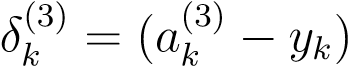
（因为第 3 层是输出层）。对于隐藏单位，您将  根据层中节点的误差项的加权平均值（*l* + 1）进行计算。

详细地说，这里是反向传播算法（如图 3 所示）。您应该在一次处理一个示例的循环中实现步骤 1 到 4。C在 c 上，您应该为 t = 1：m 实现一个 for 循环，并将下面的步骤 1-4 放在 for 循环中，*第 t 次*迭代在 *t次*训练示例 （*x*（*t*）*，y*（*t*）） 上执行计算。步骤 5 将累积的梯度除以 *m*，以获得神经网络成本函数的梯度。

1. 将输入层的值 （*a*（1）） 设置为第 *t 个*训练示例 *x*（*t*）。

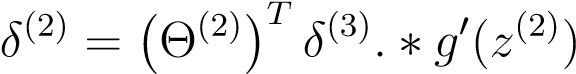
执行前馈传递（图 2），计算第 2 层和第 3 层的激活 （z（2）*、a*（2）、z（3）*、a*（3））。 请注意，您需要添加一个 +1 项，以确保层 *a*（1） 和 *a*（2）  的激活向量也包括偏置单元。在 Octave/MATLAB 中，如果 1 是列向量，则添加一个对应于 1 = [1 ; a 1].

1. 对于第 3 层（输出层）中的每个输出 unit *k*，设置

*,*

其中 *yk* ∈ {0*，*1} 表示当前训练示例是否属于类 *k* （*yk* = 1），或者它是否属于其他类 （*yk* = 0）。您可能会发现逻辑数组对此任务很有帮助（在previous 编程练习中进行了解释）。

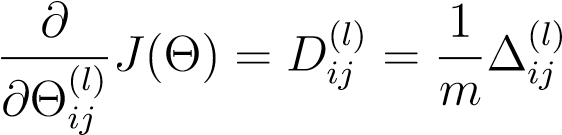
1. 对于隐藏层 *l* = 2，设置



1. 使用以下公式累积此示例中的渐变。请注意，您应该跳过或删除。在 Octave/MATLAB 中，移除对应于 delta 2 = delta 2（2：end）。

∆（*l*） = ∆（*l*） + *δ*（*l*+1）（*a*（*l*））*T*

1. 通过将累积的梯度除以以下公式，获得神经网络成本函数的（未映射）梯度：

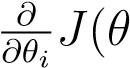


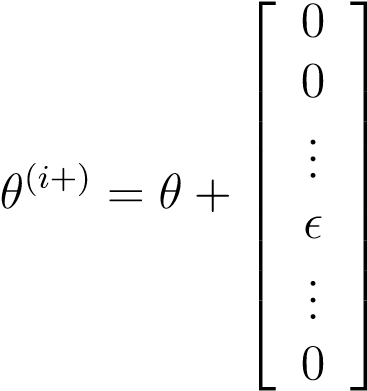
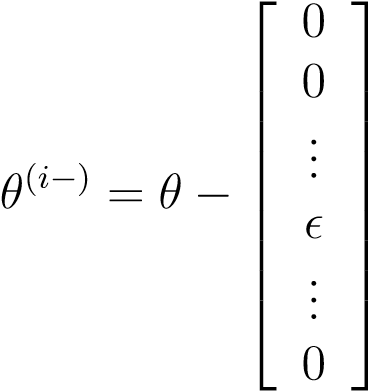
**Octave/MATLAB 提示：**只有在成功完成前馈和成本函数后，才应实现反向传播算法。在实现反向传播算法时，如果遇到维度不匹配错误（Octave/MATLAB 中的“不合格参数”错误），则使用 size 函数打印出正在使用的变量的大小通常很有用。

实现反向传播算法后，脚本 ex4.m 将继续对您的实现运行梯度检查。梯度检查将允许您增加代码正确计算梯度的信心。

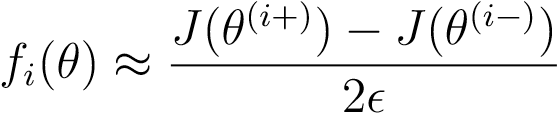
## 梯度检查

在神经网络中，您正在最小化成本函数 *J*（Θ）。要对参数执行梯度检查，您可以想象将参数 Θ（1）*，*Θ（2）  “展开”为长向量 *θ*。通过这样做，您可以将成本函数视为 *J*（*θ*），并使用以下梯度检查过程。

假设 you 有一个函数 *fi*（*θ*）， 据称它计算 ）;您想检查 *fi* 是否输出正确的导数值。

让  和 

因此，*θ*（*i*+） 与 *θ* 相同，只是其第 *i* 个元素已递增 。 类似地，*θ*（*i*−）  是第 *i* 个元素递减的相应向量。 现在，您可以通过检查每个 i 来数值验证 *fi*（*θ*） 的正确性：

*.*

这两个值应该相互近似的程度将取决于*J*的细节。但总而言之，您通常会发现上述内容的左侧和右侧将同意至少4个有效数字（通常还有更多）。

我们已经实现了该函数，用于在 computeNumericalGradient.m 中为您计算数值梯度。虽然您不需要修改文件，但我们强烈建议您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，它将创建一个小型神经网络，数据集将用于检查梯度。如果您的反向传播实现是正确的，

您应该看到小于 1e-9 的相对差异。

**实用技巧：**在执行梯度检查时，使用具有相对较少数量的输入单元和隐藏单元的小型神经网络要有效得多，因此具有相对较少的参数数量。θ 的每个维度 都需要对成本函数进行两次评估，这可能很昂贵。在函数 checkNNGradients 中，我们的代码创建了一个小型随机模型和数据集，该模型和数据集与 computeNumericalGradient 一起使用以进行梯度检查。此外，在确信梯度计算正确后，应在运行学习算法之前关闭梯度检查。

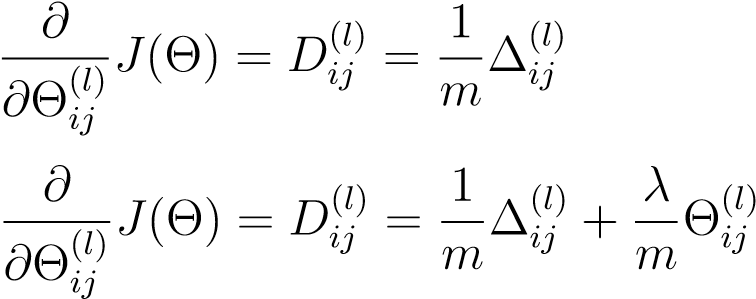
**实用技巧：** 梯度检查适用于计算成本和梯度的任何函数。具体而言，您可以使用相同的 computeNumericalGradient.m 函数来检查其他练习的梯度实现是否正确（例如，逻辑回归的成本函数）。

*一旦您的成本函数通过了（未映射的）神经网络成本函数的梯度检查，您就提交了神经网络梯度函数（反向传播）。*

## 正则化神经网络

成功实现反向传播算法后，将正则化添加到梯度。为了解释正则化，事实证明，在使用反向传播计算梯度*后*，您可以将其添加为附加项。

具体来说，在使用反向传播计算∆（*ijl*）后，您应该使用

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

 （*i*） （*l*） 

I *1.0* I *1.1* *...*

Θ（*l*） = Θ（2*和，*）0 Θ（2*l，*）1 *.*

 ... ...

有点令人困惑的是，Octave/MATLAB中的索引从1开始（对于*i*和*j*），因此Theta1（2，1）实际上对应于Θ（即，上面显示的矩阵Θ（1）  的第二行中的条目）

现在，修改在 nnCostFunction 中计算 grad 的代码，以考虑正则化。完成后，ex4.m 脚本将继续对您的实现运行梯度检查。如果您的代码正确，您应该会看到小于 1e-9 的相对差异。*您现在应该订阅您的解决方案。*

## 使用 fmincg 学习参数

在成功实现神经网络成本函数和梯度计算后，ex4.m 脚本的下一步将使用 fmincg 来学习一个好的设置参数。

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准确性。如果您的实现是正确的，您应该会看到报告的训练准确率约为95.3%（由于随机初始化，这可能会变化约1%）。通过训练神经网络进行更多迭代，可以获得更高的训练精度。我们鼓励您尝试训练神经网络进行更多迭代（例如，将 MaxIter 设置为 400），并改变正则离子参数 *λ*。通过正确的学习设置，可以使神经网络完全适合训练集。

# 可视化隐藏层

理解神经网络正在学习什么的一种方法是可视化 隐藏单元捕获的表示。非正式地说，给定一个特定的隐藏单元，可视化它计算的内容的一种方法是找到一个输入x，该x将导致它激活（即，具有激活值）。

） 接近 1）。对于您训练的神经网络，notice 表示 Θ（1） *的第* *i* 行是一个 401 维向量，表示第 *i个*隐藏单元的参数。如果我们丢弃偏置项，我们得到一个400维向量，该向量表示从每个输入像素到隐藏单元的权重。

因此，可视化隐藏单元捕获的“表示”的一个w是将这个400维向量重塑 为20×20图像并显示它。[[3]](#footnote-3)ex4.m 的下一步通过使用displayData函数来实现这 一点，它将向您显示一个具有25个单元的图像（类似于Figure 4），每个单元对应于网络中的一个隐藏单元。

在经过训练的网络中，您应该发现隐藏单元大致对应于在输入中查找笔画和其他模式的检测器。

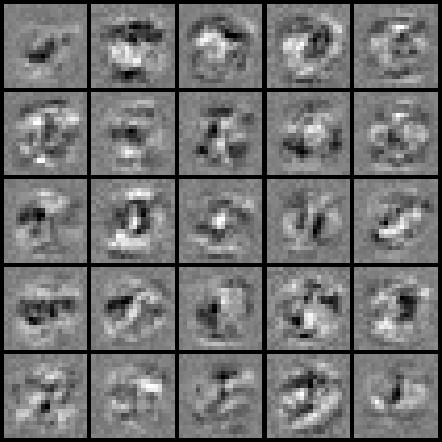


图 4：隐藏单元的可视化。

## 可选（未评分）锻炼

在练习的这一部分中，您将尝试神经网络的不同学习设置，以查看神经网络的性能如何随正则化参数 *λ* 和 训练步骤数而变化。

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。如果没有正则化，神经网络就有可能“过度拟合”训练集，使其在训练集上获得 clo se 至 100% 的准确率，但在以前从未见过的新示例上却不那么准确。您可以将正则化 *λ* 设置为较小的值，将 MaxIter 参数设置为更高的迭代次数，以便自己查看。

当您更改学习参数*λ*和MaxIter时，您还可以亲眼看到隐藏单元的可视化变化。

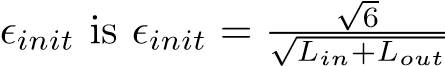
*您无需为此可选（未评分）练习提交任何解决方案。*

# 提交和评分

在完成作业的各个部分后，请务必使用 提交 功能系统将您的解决方案提交给我们的服务器。以下是本练习每个部分的评分方式的细分。

|  |  |  |
| --- | --- | --- |
| **部分** | **提交的文件** | **点** |
| 前馈和成本函数 | nnCostFunction.m | 30 积分 |
| 正则化成本函数 | nnCostFunction.m | 15 积分 |
| 乙状结肠梯度 | sigmoidGradient.m | 5 积分 |
| 神经网络 梯度 函数  （反向传播） | nnCostFunction.m | 40 积分 |
| 正则化渐变 | nnCostFunction.m | 10 积分 |
| 总积分 |  | 100 积分 |

您可以多次提交解决方案，我们将仅考虑最高分。

1. Octave 是 MATLAB 的免费替代品。对于编程练习，您可以自由使用Octave或MATLAB。 [↑](#footnote-ref-1)
2. 一个有效的选择策略是基于网络中的单位数量。一个不错的选择哪里*L在*=*sl* 和*L外*=*sl*+1是

   与 Θ 相邻的图层中的单位数(*l*). [↑](#footnote-ref-2)
3. 事实证明，这等效于找到为隐藏单元提供最高激活的输入，给定对输入的“范数”约束（即，k*x*k2≤1). [↑](#footnote-ref-3)