BP 神经网络

介绍

在本实验中，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的任务。

本实验中包含的文件

ex4.py-帮助你完成实验的 Python 脚本

ex4data1.mat - 手写数字训练集

ex4weights.mat - 神经网络参数

displayData.py -可视化数据集函数

sigmoid.py – Sigmoid 函数

computeNumericalGradient.py -计算数值梯度函数

checkNNGradients.py - 梯度验证函数

debugInitializeWeights.py–初始化权重

predict.py– 预测函数

[*]SigmoidGradient.py--计算 Sigmoid 函数的梯度

[*]randInitializeWeights.py-随机初始化权重

[*] nnCostFunction.py-BP 神经网络损失函数

* 代表你需要完成的文件

1 神经网络

在这个实验中，你将实现反向传播算法来学习神经网络的参数。

1.1 数据可视化

在 ex4.py 的第一部分代码将加载数据并通过调用函数 displayData 将其显示在一个二维图形上(图 1)。

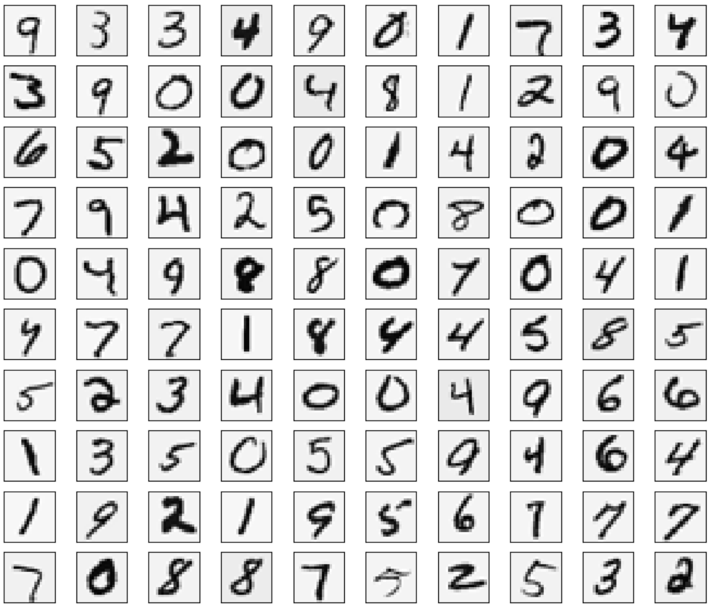

图 1:部分数据集

在 ex3data1.mat 中有 5000 个训练数据，其中每个训练数据都是 20 像素× 20 像素灰度图像的数字。每个像素由一个浮点数表示，表示该位置的灰度强度。20 × 20 像素矩阵被“展开”成一个 400 维的矢量。这些训练数据中的一个对应数据矩阵 X 中的一行，这就得到了一个 5000 × 400 的矩阵 X，其中每一行都是一个手写数字图像的训练数据。：

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

训练集的第二部分是一个 5000 维的向量 y，它包含训练集的标签。将数字 0 映射为值 10。因此，数字“0”被标记为“10”，数字“1”到“9”按照自然顺序被标记为“1”到“9”。

1.2 模型表示

BP 神经网络如图 2 所示。它有 3 层即输入层，隐藏层和输出层。我们的输入是数字图像的像素值。因为图像的尺寸是 20 × 20，这给了我们 400 个输入层单元(不包括额外偏置单元)。训练数据将被 ex4.py 加载到变量 X 和 y 中。

在文件中提供了一个训练好的模型存储在 ex4weights.mat 中，将由 ex4.py 加

载变成 Theta1 和 Theta2。参数的形状对应第二层有 25 个单元和 10 个输出单元(对应于 10 个数字类)的神经网络。

```
# Load the weights into variables Theta1 and Theta2
data = loadmat('ex4weights.mat')
Theta1, Theta2 = data['Theta1'], data['Theta2']
# Unroll parameters
```

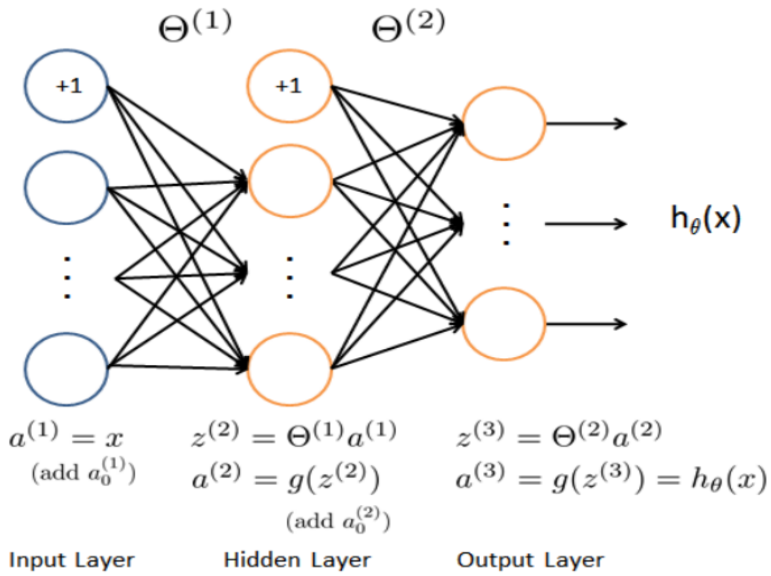

Figure 2: Neural network model.

1.3 前馈与代价函数

现在你将实现神经网络的代价函数和梯度。首先，在 nnCostFunction.py 中完成代码来返回代价。

神经网络的代价函数(没有正则化)是：

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right],$$

其中$h_\theta(x^{(i)})$的计算如图 2 所示，K = 10 是可能的标签总数。注意，$h_\theta\left(x^{(i)}\right)_k = a_k^{(3)}$，k 是第 k 个输出单元的激活值(输出值)。另外，虽然原始的标签(在变量 y 中)是 1,2，…， 10，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，因此需要将不同值转化为相应的变量:

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

例如，如果$x^{(5)}$是数字 5 的图像，那么相应的$y_{(i)}$(你应该使用的代价函数)应该是一个 10 维向量，$y_{(5)} = 1$，其他元素等于 0。

你应该实现前馈计算，计算每个例子 i 的$h_\theta(x^{(i)})$，并对所有例子求和。您的代码还应该适用于任何大小的数据集和任何数量的标签(你可以假设至少有 K≥3 个标签)。

1.4 正则化的代价函数

正则化神经网络的代价函数为：

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

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，你的代码应该可以处理任意数量的输入单元、隐藏单元和输出单元。为了使公式简单易懂，我们明确列出了上面的 Θ(1)和 Θ(2)的索引，但你的代码应该适用于任何大小输入。

注意，你可以首先使用现有的 nnCostFunction.py 计算非正则的代价函数 J,然后再加上正则化项的代价。

一旦完善了代码，ex4.py 将调用 nnCostFunction.py 使用加载的参数集的 Theta1 和 Theta2，λ=1。您应该可以看到代价约为 0.383770。

2 反向传播

在练习的这一部分中，您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.py，以便返回 grad 的适当值。一旦你计算了梯度，你就可以通过使用 fmincg 等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(非正则)神经网络参数的梯度。在验证了非正则情况下的梯度计算是正确的之后，你将实现正则神经网络的梯度。

2.1 Sigmoid 梯度

Sigmoid 函数的梯度可以计算为：

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1 - g(z))$$

其中：

$$\mathrm{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}.$$

2.2 随机初始化

在训练神经网络时，随机初始化对打破对称是非常重要的。一种有效的随机初始化策略是在$[\epsilon_{init}, \epsilon_{init}]$范围内均匀随机选取 Θ(l)的值。你可以用$\epsilon_{init}$= 0.12，这个范围的值可以确保参数保持在较小的范围内，并使学习更有效。

2.3 反向传播

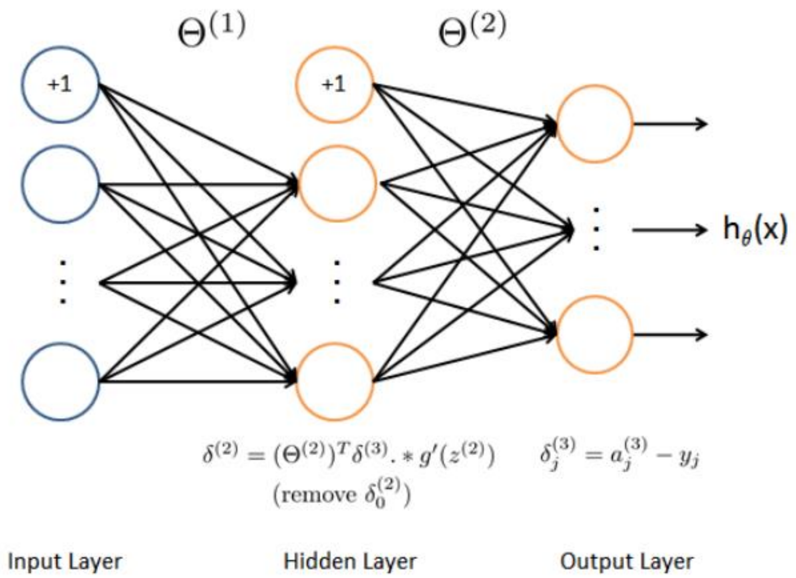

Figure 3: Backpropagation Updates.

现在，您将实现反向传播算法。回想一下反向传播算法背后是这样的，给定一个训练数据(x^t, y^t)，我们首先运行一个“正向传递”来计算整个网络中的所有激活节点的值，包括假设$h_{\Theta(x)}$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”δ_j^l，以度量该节点对我们输出中的误差“负责”的程度。

具体来说，这里是反向传播算法(也在图 3 中描述)。你应该在一个循环中实现步骤 1 到 4，每次处理一个数据。具体来说，你应该为 t = 1:m 实现一个 for 循环，并将步骤 1-4 放在 for 循环中，第 t 次迭代对第 t 个训练数据$(x^{(t)}, y^{(t)})$执行计算。步骤 5 将累积的梯度除以 m，得到神经网络代价函数的梯度。

1. 设置输入层的值$(a^{(1)})$为第 t 个训练数据$x^{(t)}$。执行前馈传递(图 2)，计算二层和三层的激活值(z(2)， a(2)， z(3)， a(3))。注意，你需要在 a 中添加一列，以确保层 a(1)和层 a(2)的激活向量也包括偏置单元。在 Python 中，如果 X 是一个列向量，加一列值全为 1 的代码为：

```
np.insert(X, 0, values=np.ones(m), axis=1)
```

2. 对于第 3 层(输出层)中的每个输出单元 k，设置

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

其中$y_k \in \{0,1\}$表示当前的训练示例是否属于类k $(y_k = 1)$，或者是否属于其它的类 $(y_k = 0)$。

3.对于隐藏层$l = 2$，设置

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

4. 使用下面的公式计算累计梯度。

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. 将累积梯度除以 m，得到神经网络代价函数的(非正则化)梯度:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

在实现了反向传播算法之后，脚本 ex4.py 将继续在这基础上运行梯度检查。梯度检查可以让你更加确信你的代码计算的梯度是正确的。

2.4 梯度检查

在神经网络中，最小化代价函数 J(Θ)。为了对参数进行梯度检查，可以想象将参数 Θ(1)， Θ(2)“展开”到一个长矢量 θ 中。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程序。

假设有一个函数$f_i(\theta)$计算$\frac{\partial}{\partial\theta_i}J(\theta)$;你想检查$f_i$是否输出正确的导数值

$$\theta^{(i+)} = \theta + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix} \quad \text{and} \quad \theta^{(i-)} = \theta - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \epsilon \\ \vdots \\ 0 \end{bmatrix}$$

$\theta^{(i)}$和θ相似除了第 i 个元素加了ϵ。类似地，$\theta^{(i-)}$是第 i 个元素减ϵ的对应向量。现在，你可以通过检查每个 i，数值验证$f_i(\theta)$的正确性:

$$f_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon}.$$

这两个值相互近似的程度取决于J的细节。但假设呢$\epsilon = 10^{-4}$，你会发现上面的左边和右边至少有 4 个有效数字(通常更多)。

我们在 computeNumericalGradient.py 中实现了计算数值梯度的函数。虽然你不需要修改该文件，但我们强烈建议您看一看代码，以理解它是如何工作的。在 ex4.py 的下一步，它将运行提供的函数 checkNNGradients。将创建一个小型的神经网络和数据集，用于检查你的梯度。如果你的反向传播实现是正确的，误差会小于 1e-9。

2.5 正则化神经网络

在你成功地实现了反向传播算法后，你将添加正则化的梯度。为了考虑正则化，你可以在使用反向传播计算梯度后添加这个附加项。

具体地说，在使用反向传播计算出$\Delta_{ij}^{(l)}$之后，你应该添加正则化使用：

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

请注意，您不应该正则化$\Theta^{(l)}$的第一列，这是偏置项。此外，在参数$\Theta_{ij}^{(l)}$中，

i 从 1 开始索引，j 从 0 开始索引。因此,

$$\Theta^{(l)} = \begin{bmatrix} \Theta_{1,0}^{(i)} & \Theta_{1,1}^{(l)} & \cdots \\ \Theta_{2,0}^{(i)} & \Theta_{2,1}^{(l)} & \\ \vdots & & \ddots \end{bmatrix}.$$

现在修改你在 nnCostFunction.py 中计算 grad 的代码，以考虑正则化。在你完成之后，脚本 ex4.py 将继续在你的实现的基础上运行梯度检查。如果你的代码是正确的，误差会小于 1e-9。

2.6 使用共轭梯度法学习参数

在成功实现神经网络代价函数和梯度计算后，下一步的 ex4.py 将使用 fmin_cg 学习一个合适的参数。

训练结束后，ex.py4 脚本将通过计算正确示例的百分比来报告分类器的训练准确性。如果你的实现是正确的，那么你应该会看到报告的训练准确率约为 95.3%(由于随机初始化，这可能会有大约 1%的差异)。通过对神经网络进行更多的迭代训练，可以获得更高的训练精度。我们鼓励您尝试为更多的迭代训练神经网络(例如，设置 MaxIter 为 100)，并改变正则化参数 λ。

3 可视化隐藏层

要理解神经网络正在学习什么，一种方法是将隐藏单元捕捉到的表征形象化。给定 12 个特定的隐藏单元，可视化计算结果的一种方法是找到一个输入 x 将它激活(即使激活值($a_i^{(l)}$)接近 1)。对于你训练的神经网络，请注意$\Theta^{(1)}$的第 i 行是一个 401 维向量，表示第 i 个隐藏单元的参数。如果我们放弃偏差项，我们得到一个 400 维的向量，它表示从每个输入像素到隐藏单元的权重。

因此，将隐藏单元捕获的“表征”可视化的一种方法是将这个 400 维向量重塑为一个 20 × 20 的图像并显示它。ex4.py 的下一步。通过使用 displayData 函数来实现这一点，它将显示一个图像(类似于图 4)，其中有 25 个单元，每个单元对应于网络中的一个隐藏单元。

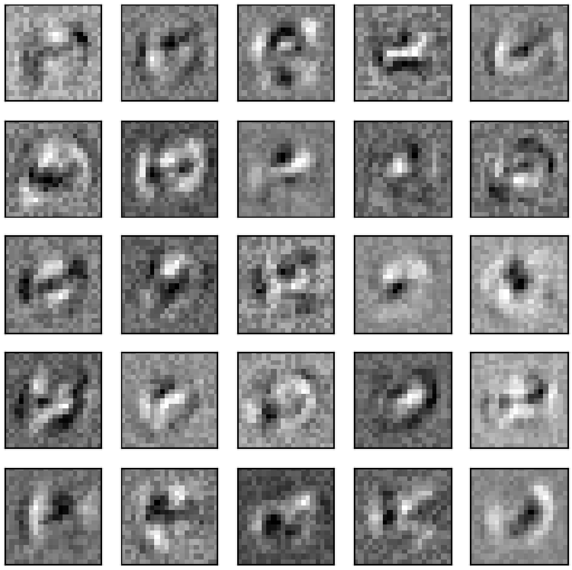

图 4:隐含单元的可视化

3.1 改变正则化参数 λ 和训练次数 MaxIter

在练习的这一部分，你将尝试不同的神经网络学习设置，看看神经网络的性能如何随正则化参数 λ 和训练步骤的数量而变化(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“过拟合”一个训练集，从而获得接近 100%的准确性训练集，但在以前没有见过的新数据集上表现就不那么好了。你可以将正则化 λ 设置为较小的值，并将 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，你也能看到隐藏单元的可视化变化。