九、神经网络的学习(Neural Networks: Learning)
9.1 代价函数
9.2 反向传播算法(1)
9.3 反向传播算法(2)
9.4 反向传播算法的直观理解
9.5 神经网络与反向传播推导与总结
9.5 梯度检验
9.6 随机初始化
9.7 综合起来
9.8 自主驾驶

九、神经网络的学习(Neural Networks: Learning)

9.1 代价函数

参考视频: 9 - 1 - Cost Function (7 min).mkv

首先引入一些便于稍后讨论的新标记方法：

1. 假设神经网络的训练样本有m个，每个包含一组输入x和一组输出信号y
2. L表示神经网络层数
3. S_l表示第l层的**neuron**个数，不包括偏差单元
4. S_L代表最后一层中处理单元的个数 如$S_L = 4$

将神经网络的分类定义为两种情况：二类分类和多类分类，

- 二类分类 **Binary classification**： $S_L = 1, y = 0\, or\, 1$ 表示哪一类
- K -类分类 **Multi-class classification**： $S_L = K, y_i = 1$ 表示分到第 i 类； $(k > 2)$

我们回顾**正则化的逻辑回归模型中的代价函数**为：

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + \left(1 - y^{(i)}\right) log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}{\theta_j}^2$$

在逻辑回归中，我们只有一个输出变量，又称标量**scalar**，也只有一个因变量y.

但是在神经网络中，我们可以有很多输出变量，我们的$h_\theta(x)$**是一个维度为**K**的向量**，$(h_\theta(x))_i = i^{th}\text{output}$ 表示的是选择了神经网络输出向量的第i个元素。我们训练集中的因变量也是同样维度的一个向量，因此我们的代价函数会比逻辑回归更加复杂一些:

$$h_\theta(x) \in \mathbb{R}^K \qquad (h_\theta(x))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k} {y_k}^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1 - y_k^{(i)}) \log\left(1 - \left(h_\Theta(x^{(i)})\right)\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

- 函数解释：
  1. 这个看起来复杂很多的代价函数背后的思想还是一样的，我们希望通过代价函数来观察算法预测的结果与真实情况的误差有多大，唯一不同的是，对于每一行特征，我们都会给出K个预测，基本上我们可以利用循环，对每一行特征都预测K个不同结果，然后在利用循环在K个预测中选择可能性最高的一个，将其与y中的实际数据进行比较。

2. 在函数的第一部分中我们额外添加了求出各个节点的代价函数的求和，
3. 在正则化部分，是排除了每一层θ_0后，每一层的θ 矩阵的和。在当前Θ矩阵中
   - 列数 i = 我们当前层中的节点数 （包括偏差单元）: 循环 i 则循环所有的列，由该层（ s_l 层）的激活单元数所决定。
   - 行数 j = 下一层的节点数 （不包括偏置单元）: 循环 j 循环所有的行（由 s_{l+1} 层的激活单元数决定）。
4. 综合即，$h_\theta(x)$与真实值之间的距 离为每个样本-每个类输出的加和，对参数进行**regularization**的**bias**项处理所有参数的平方和。

9.2 反向传播算法(1)

参考视频: 9 - 2 - Backpropagation Algorithm (12 min).mkv

H3 反向传播是神经网络中用于最小化代价函数的一个方法，类似于在logistic回归和线性回归中使用的梯度下降，目标都是计算

$$\min_{\Theta} J(\theta)$$

也就是说，我们需要找到一个最佳的参数来最小化J，因此我们需要计算：

1. $J(\Theta)$
2. $J(\Theta)$ 的偏导数 $\frac{\partial}{\partial\Theta_{i,j}^{(l)}}J(\Theta)$

现在，为了计算代价函数的偏导数，我们需要采用一种反向传播算法**Backpropagation Algorithm**。

以一个例子来说明反向传播算法。

假设我们的训练集只有一个样本(x,y)，我们的神经网络是一个四层的神经网络，其中 $K=4,\quad S_L=4,\quad L=4$:

- 前向传播算法

  使用前向传播来计算一下在给定输入的时候，假设函数是否会真的输出结果。具体地说，这里的$a^{(1)}$就是第一层的激励值，也就是输入层所在的地方，所以设定它为x。$a^{(2)}$是隐藏层也就是第二层的激励值,经过一个S型激励函数$g(z)$，同时还会增加一个偏差单元$a_0^{(2)}$,然后可以得出后续的$a^{(3)}, a^{(4)}$。

- 反向传播算法

  为了计算导数项，将采用反向传播的算法。反向传播算法从直观上说，就是对每一个节点

  $\delta_j^{(l)}$ 代表的是第l层的第j个节点的误差

  $a_j^{(l)}$ 代表的是第l层第j个单元的激励值

所以这个δ在某种程度上就捕捉到了在这 个神经节点的激励值的误差，所以我们可能希望这个节点激励值稍微不一样。

如上图，用这个有四层的神经网络结构做例子：

- 对于 **第四层**

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$
$$\delta_j^{(4)} = (h_\theta(x))_j - y_j$$

即第四层第j个节点激励值与实际值的误差，用向量方式表示就是

$$\delta^{(4)} = a^{(4)} - y$$

其中δ, a, y向量的维数就是输出单元的个数。

- 对于**第三层**，实际上是

$$\delta_j^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .*g'(z^{(3)})$$

  - 解释
    1. .*$点乘指的是两个向量元素间对应相乘。
    2. $g'(z^{(3)})$是对激励函数g在输入值为$z(3)$的时候所求的导数。

$$g'(z^{(3)}) = a^{(3)} .*(1 - a^{(3)})$$

    - $a^{(3)}$ 是一个对那一层的激励向量
    - 1是以1为元素的向量（这一块没看懂）。

- 对于**第二层**，实际上是

$$\delta_j^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .*g'(z^{(2)})$$
$$g'(z^{(2)}) = a^{(2)} .*(1 - a^{(2)})$$

- 对于**第一层**

  是没有$\delta_j^{(1)}$项，因为第一层是输入层，即实际的训练集观察到的，因此不会有误差。

反向传播算法也就是首先计算最后一层的误差，然后再一层一层反向求出各层的误差，直到倒数第二层。

利用反向传播算法，在忽略正则化项时，我们可以得出

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

通过反向传播，可以计算出这些δ项进而计算出所有参数的偏导数项。

9.3 反向传播算法(2)

当我们有一个**非常大**的训练样本时如何来计算关于这些参数的偏导数，使用下面的算法：

H3

1. 假设有一组训练集$\{(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})\}$,
2. 假设对于所有的l, i, j都有$\Delta_{ij}^{(l)} = 0$，这个$\Delta_{ij}^{(l)}$实际上是用来计算

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

   这些Δ会被作为累加项慢慢的增加以算出这些偏导数。
3. 接下来先循环我们的训练集 for $i = 1$ to m
   1. 设定$a^{(1)} = x^{(i)}$,
      - $a^{(1)}$ 输入层的激励函数

- $x^{(i)}$ 第i个训练样本的输入值

2. 使用**前向传播算法**来计算所有的$a^{(l)}$直到最后一层L
3. 使用这个样本的输出值$y^{(i)}$来计算这个输出值对应的**误差项**$\delta^{(L)}$，即假设输出减去目标输出
4. 使用**反向传播算法**来计算$\delta^{(L-1)}, \delta^{(L-2)}, \delta^{(L-3)}, \ldots, \delta^{(2)}$,注意没有$\delta_j^{(1)}$项
5. 最后将大写的Δ来累计我们在前面写好的偏导数项，即：

$$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

如果把Δ_{ij}看作一个矩阵，i, j代表矩阵中的位置，那么该等式可以用向量的方式表示为

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

6. 结束循环

4. 根据j是否等于0来分两种情况计算

$$\begin{aligned} D_{ij}^{(l)} &:= \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} \qquad \text{if } j \neq 0 \\ D_{ij}^{(l)} &:= \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{if } j = 0 \end{aligned}$$

当$j = 0$时对应偏差项，因此没有加上额外的标准化项。

5. 这样，得到的就是代价函数对每一个参数的偏导数

$$\frac{\partial}{\partial\Theta_{i,j}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

这样就可以使用梯度下降法或是其他高级算法了。

9.4 反向传播算法的直观理解

参考视频: 9 - 3 - Backpropagation Intuition (13 min).mkv

为了更好地理解反向传播算法，我们再来仔细研究一下前向传播的原理。

H3

如上图所示的神经网络，输入层，隐藏层各有两个单元（不包含偏差单元），输出层有一个单元。

反向传播算法做的是：

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{k} {y_k}^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1-y_k^{(i)}) \log\Big(1-\left(h_\Theta(x^{(i)})\right)\Big)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

如果我们只考虑**一个简单样本**$x^{(i)}, y^{(i)}$且只有一个输出单元的情况，即$k=1$，且忽略正则项($\lambda=0$)那么代价函数则变成：

$$\text{cost}(i) = y^{(i)} \log\Big(h_\Theta(x^{(i)})\Big) + \Big(1-y^{(i)}\Big) \log\Big(h_\Theta(x^{(i)})\Big)$$

我们可以认为$cost(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$

- $\delta_j^{(l)}$ = 'error' of cost for $a_j^{(l)}$ (unit j in layer l) 理解如下：
  1. $\delta_j^{(l)}$ 相当于是第 l 层的第 j 单元中得到的激活项的“误差”，即”正确“的 $a_j^{(l)}$ 与计算得到的 $a_j^{(l)}$ 的差。
  2. 又因为 $a_j^{(l)} = g(z^{(l)})$ ，（g 为sigmoid函数）。我们可以想象 $\delta_j^{(l)}$ 为函数求导时迈出的那一丁点微分。
  3. 更准确的说 $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} cost(i)$

由于函数的导数即是其切线的斜率，因此其切线越陡则说明计算的误差越大。每一个神经元的误差都与后面连接的神经元有关。

9.5 神经网络与反向传播推导与总结

H3

$$x=\begin{bmatrix}0.05\\0.1\end{bmatrix} \quad y=\begin{bmatrix}0.01\\0.99\end{bmatrix}$$

(一)前向传播

$$h_1 = W_{11}\cdot x_1 + W_{12}\cdot x_2 + b_1^{(1)} = 0.15\times0.05 + 0.2\times0.1 + 0.35 = 0.3775$$

$$a_1 = g(h_1) = \frac{1}{1+e^{-h_1}} = 0.5933$$

$$h_3 = W_{31}\cdot a_1 + W_{32}\cdot a_2 + b_1^{(2)} = 1.1059$$

$$a_3 = g(h_3) = \frac{1}{1+e^{-h_3}} = 0.751$$

(二) 用矩阵进行前向传播

$$\underset{3\times1}{X=\begin{pmatrix}1\\0.05\\0.1\end{pmatrix}} \quad \underset{2\times3}{W_1=\begin{pmatrix}b_1^{(1)} & W_{11} & W_{12}\\ b_2^{(1)} & W_{21} & W_{22}\end{pmatrix}} = \begin{pmatrix}0.35 & 0.15 & 0.2\\ 0.35 & 0.25 & 0.3\end{pmatrix}$$ 权重矩阵

$$\underset{2\times1}{H=\begin{pmatrix}h_1\\h_2\end{pmatrix}} = W_1\cdot X = \begin{pmatrix}0.3775\\0.5925\end{pmatrix} \quad A=\begin{pmatrix}a_1\\a_2\end{pmatrix} = \begin{pmatrix}0.5933\\0.5969\end{pmatrix}$$ 激活矩阵

$$A'=\begin{pmatrix}1\\a_1\\a_2\end{pmatrix} \quad h=\begin{pmatrix}h_3\\h_4\end{pmatrix} = W_2\cdot A' \quad a=\begin{pmatrix}a_3\\a_4\end{pmatrix} = \begin{pmatrix}0.751\\0.773\end{pmatrix}$$

（三）铺垫知识

① 链式法则:

$x=g(s)$　　$y=h(s)$　　$z=k(x,y)$　　求 $\frac{dz}{ds}$

则 $\frac{dz}{ds}=\frac{dz}{dx}\cdot\frac{dx}{ds}+\frac{dz}{dy}\cdot\frac{dy}{ds}$

例：$z=e^{u}\sin v$　　$u=xy$　　$v=x+y$　　求 $\frac{dz}{dx}$，$\frac{dz}{dy}$

则 $\frac{dz}{dx}=\frac{dz}{du}\cdot\frac{du}{dx}+\frac{dz}{dv}\cdot\frac{dv}{dx}=e^{u}\cdot\sin v\cdot y-e^{u}\cos v$

$\frac{dz}{dy}=\frac{dz}{du}\cdot\frac{du}{dy}+\frac{dz}{dv}\cdot\frac{dv}{dy}=e^{u}\cdot\sin v\cdot x-e^{u}\cos v$

② 目的

利用梯度下降法更新 w, b

$$\begin{cases} w := w-\alpha\cdot\frac{\partial J}{\partial w} \\ b := b-\alpha\cdot\frac{\partial J}{\partial b} \end{cases}$$

求 $J(\theta)$ 和 两个偏导

假设：
$$\begin{cases} J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 \\ J(\theta)=\frac{1}{2}(h_\theta(x)-y)^2 \end{cases}$$
----平方损失

计算损失：$J_1=\frac{1}{2}(a_3-y^{(1)})^2=\frac{1}{2}(0.7514-0.01)^2=0.2745$

$J_2=\frac{1}{2}(a_4-y^{(2)})^2=\frac{1}{2}(0.773-0.99)^2=0.0235$

$J(\theta)=J_1+J_2=0.298$

（四）反向传播的一般计算

① 更新输出层到隐藏层的 W 与 b

1) 更新 W_{31}, W_{41}, W_{32}, W_{42}.

$$J(\theta) = J_1 + J_2 = \underbrace{\tfrac{1}{2}(a_3 - y^{(1)})^2}_{J_1} + \underbrace{\tfrac{1}{2}(a_4 - y^{(2)})^2}_{J_2}$$

$$\frac{\partial J}{\partial W_{31}} = \frac{\partial J}{\partial J_1} \cdot \frac{\partial J_1}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial W_{31}}$$

$$= 1 \cdot (a_3 - y^{(1)}) \cdot [a_3 \cdot (1 - a_3)] \cdot a_1 = 0.0822$$

$h_3 = W_{31} a_1 + W_{32} a_2 + b_1^{(2)} \cdot 1$

$a_3 = g(h_3) = \frac{1}{1+e^{-h_3}}$

$J(\theta) \begin{cases} J_1 - a_3 - h_3 - W_{31} \\ J_2 - a_4 - h_4 \quad \cdots \end{cases}$

使用梯度下降： $\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$ 设 $\alpha = 0.5$ 学习率

$$W_{31} = W_{31} - \alpha \cdot \frac{\partial J}{\partial W_{31}} = 0.4 - (0.5 \cdot 0.0822) = 0.3589$$

同理，可计算得到 W_{41}, W_{32}, W_{42}.

2) 更新偏置 $b_1^{(2)}$, $b_2^{(2)}$

$$J(\theta) = J_1 + J_2$$

$$\frac{\partial J}{\partial b_1^{(2)}} = \frac{\partial J}{\partial J_1} \cdot \frac{\partial J_1}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial b_1^{(2)}} = 1 \cdot (a_3 - y^{(1)}) \cdot [a_3 \cdot (1 - a_3)] \cdot 1 = 0.1385$$

$$b_1^{(2)} = b_1^{(2)} - \alpha \cdot \frac{\partial J}{\partial b_1^{(2)}} = 0.52 - (0.5 \times 0.1385) = 0.5307$$

同理，可计算得到 $b_2^{(2)}$

$$h_1 = W_{11} x_1 + W_{12} x_2 + b_1^{(1)} \cdot 1$$

$$a_1 = g(h_1) = \frac{1}{1+e^{-h_1}}$$

$$J(\theta) \begin{cases} J_1 - a_3 - h_3 - a_1 - h_1 - W_{11} \\ J_2 - a_4 - h_4 \end{cases}$$

② 更新隐藏层到输入层 W 与 b

1) 更新 $W_{11}, W_{21}, W_{12}, W_{22}$.

$$\frac{dJ}{dW_{11}} = \boxed{\frac{dJ}{da_1}} \cdot \frac{da_1}{dh_1} \cdot \frac{dh_1}{dW_{11}}$$

$$\Downarrow$$

$$\frac{dJ}{da_1} = \left(\frac{dJ}{dJ_1} \cdot \frac{dJ_1}{da_3} \cdot \frac{da_3}{dh_3} \cdot \frac{dh_3}{da_1}\right) + \left(\frac{dJ}{dJ_2} \cdot \frac{dJ_2}{dJ_4} \cdot \frac{dJ_4}{dh_4} \cdot \frac{dh_4}{da_1}\right)$$

$$= \left(1 \cdot (a_3 - y^{(1)}) \cdot [a_3 \cdot (1-a_3)] \cdot W_{31}\right) + \left(1 \cdot (a_4 - y^{(2)}) \cdot [a_4 \cdot (1-a_4)] \cdot W_{41}\right)$$

$$= 0.03636$$

$$\frac{dJ}{dW_{11}} = \frac{dJ}{da_1} \cdot [a_1 \cdot (1-a_1)] \cdot x$$

$$= (0.03636) \cdot (0.241295) \cdot (0.05) = 0.0004387$$

使用梯度下降：$W_{11} = W_{11} - \alpha \frac{dJ}{dW_{11}} = 0.15 - (0.5 \times 0.0004387)$

$$= 0.1498$$

同理，可计算得到 W_{21}, W_{12}, W_{22}.

2) 更新偏置 $b_1^{(1)}, b_2^{(1)}$.

$$J(\theta) = J_1 + J_2$$

$$\frac{dJ}{db_1^{(1)}} = \frac{dJ}{da_1} \cdot \frac{da_1}{dh_1} \cdot \frac{dh_1}{db_1^{(1)}} = \frac{dJ}{da_1} \cdot [a_1 \cdot (1-a_1)] \cdot 1 = 0.0088$$

$$b_1^{(1)} = b_1^{(1)} - \alpha \cdot \frac{dJ}{db_1^{(1)}} = 0.35 - (0.5 \times 0.0088) = 0.3456$$

同理，可计算得到 $b_2^{(1)}$.

（五）反向传播的矩阵方法

定义：误差项：在这个神经节点的激励值的误差

$$\delta^{(2)}=\frac{dJ}{dh} \qquad \delta^{(1)}=\frac{dJ}{dH}$$

$$H=\begin{pmatrix}h_1\\h_2\end{pmatrix} \quad h=\begin{pmatrix}h_3\\h_4\end{pmatrix}$$

$$A=\begin{pmatrix}a_1\\a_2\end{pmatrix} \quad a=\begin{pmatrix}a_3\\a_4\end{pmatrix}$$

$$J(\theta)=\frac{1}{2}(a-y)^2$$

$$a=\begin{pmatrix}a_3\\a_4\end{pmatrix} \quad y=\begin{pmatrix}0.01\\0.99\end{pmatrix}$$

S型函数

① 计算误差项 $\delta^{(1)}$，$\delta^{(2)}$

1) $\delta^{(2)}=\frac{dJ}{dh}=\frac{dJ}{da}\cdot\frac{da}{dh}$

$$=(a-y)\cdot[a\cdot(1-a)]$$

$g'(a)$

$$=\left[\begin{pmatrix}0.751\\0.773\end{pmatrix}-\begin{pmatrix}0.01\\0.99\end{pmatrix}\right]\odot\left[\begin{pmatrix}0.751\\0.773\end{pmatrix}\odot\left[E-\begin{pmatrix}0.751\\0.773\end{pmatrix}\right]\right]$$

$$=\begin{pmatrix}0.74\\-0.217\end{pmatrix}\odot\begin{pmatrix}0.249\\0.129\end{pmatrix}=\begin{pmatrix}0.1386\\-0.0386\end{pmatrix}$$

$\vec{a}=(x_1,y_1)$

$\vec{b}=(x_2,y_2)$

$\vec{a}\cdot\vec{b}=x_1x_2+y_1\cdot y_2$

点乘：由于 (1×2) 不能与 (1×2) 叉乘

2) $\delta^{(1)} = \frac{dJ}{dH} = \underbrace{\boxed{\frac{dJ}{dh}}}_{\delta^{(2)}} \cdot \frac{dh}{dA} \cdot \frac{dA}{dH}$

$h = W_2 \cdot A + b^{(2)} = \begin{pmatrix} h_3 \\ h_4 \end{pmatrix}$

$A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$

矩阵求导 $\frac{dh}{dA} = W_2^T = \begin{pmatrix} W_{31} & W_{32} \\ W_{41} & W_{42} \end{pmatrix}^T = \begin{pmatrix} 0.4 & 0.5 \\ 0.45 & 0.55 \end{pmatrix}$

$\frac{dA}{dH} = A(1-A) = \begin{pmatrix} 0.5933 \\ 0.5968 \end{pmatrix} \odot \left[E - \begin{pmatrix} 0.5933 \\ 0.5968 \end{pmatrix} \right] = \begin{pmatrix} 0.2413 \\ 0.2406 \end{pmatrix}$

$\therefore \delta^{(1)} = \underset{2\times1}{\begin{pmatrix} 0.1386 \\ -0.0386 \end{pmatrix}} \cdot \underset{2\times2}{\begin{pmatrix} 0.4 & 0.5 \\ 0.45 & 0.55 \end{pmatrix}} \cdot \underset{2\times1}{\begin{pmatrix} 0.2413 \\ 0.2406 \end{pmatrix}}$

交换

$= \underset{2\times1}{(W_2^T \cdot \delta^{(2)})} \odot A(1-A) = \begin{pmatrix} 0.0087 \\ 0.0099 \end{pmatrix}$

② 更新输出层到隐藏层的 W 与 b

$h = W_2 \cdot A + b^{(2)} = \begin{pmatrix} h_3 \\ h_4 \end{pmatrix}$

1) 更新 W_2

$\frac{dJ}{dW_2} = \underbrace{\boxed{\frac{dJ}{dh}}}_{\delta^{(2)}} \cdot \frac{dh}{dW_2} = \delta^{(2)} \cdot A^T = \underset{2\times1}{\begin{pmatrix} 0.1386 \\ -0.0386 \end{pmatrix}} \cdot \underset{1\times2}{\begin{pmatrix} 0.5933 & 0.5968 \end{pmatrix}}$

$= \begin{pmatrix} 0.0822 & 0.0827 \\ -0.0229 & -0.023 \end{pmatrix}$

使用梯度下降：$W_2 = W_2 - \alpha \frac{dJ}{dW_2} = \begin{pmatrix} 0.3581 & 0.4086 \\ 0.5115 & 0.5615 \end{pmatrix}$

2) 更新偏置 $b^{(2)}$

$$\frac{dJ}{db^{(2)}} = \frac{dJ}{dh} \cdot \frac{dh}{db^{(2)}} = \delta^{(2)} \cdot E = \begin{pmatrix} 0.1386 \\ -0.0386 \end{pmatrix}$$

使用梯度下降：$b^{(2)} = b^{(2)} - \alpha \frac{dJ}{db^{(2)}} = \begin{pmatrix} 0.5307 \\ 0.6193 \end{pmatrix}$

综上：① $\delta^{(2)} = \frac{dJ}{da} \odot g'(a)$

$$\delta^{(1)} = (W_2^T \cdot \delta^{(2)}) \odot g'(A)$$

② $\frac{dJ}{dW_2} = \delta^{(2)} \cdot A^T$

$$\frac{dJ}{db^{(2)}} = \delta^{(2)}$$

③ 更新隐藏层到输入层 W 与 b

$$H = W_1 \cdot X + b^{(1)} = \begin{pmatrix} h_1 \\ h_2 \end{pmatrix}$$

1) 更新 W_1

$$\frac{\partial J}{\partial W_1} = \underbrace{\boxed{\frac{\partial J}{\partial H}}}_{\delta^{(1)}} \cdot \frac{\partial H}{\partial W_1} \qquad \frac{\partial H}{\partial W_1} = X^T$$

$$= \delta^{(1)} \cdot X^T = \begin{pmatrix} 0.0087 \\ 0.0099 \end{pmatrix} \cdot \begin{pmatrix} 0.05 & 1 \end{pmatrix} = \begin{pmatrix} 0.000425 & 0.00087 \\ 0.000495 & 0.00099 \end{pmatrix}$$

使用梯度下降：

$$W_1 = W_1 - \alpha \frac{\partial J}{\partial W_1} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} - \alpha \frac{\partial J}{\partial W_1}$$

$$= \begin{pmatrix} 0.15 & 0.2 \\ 0.25 & 0.3 \end{pmatrix} - 0.5 \times \begin{pmatrix} 0.000425 & 0.00087 \\ 0.000495 & 0.00099 \end{pmatrix}$$

$$= \begin{pmatrix} 0.1498 & 0.1996 \\ 0.2498 & 0.2995 \end{pmatrix}$$

2) 更新偏置 $b^{(1)}$

$$\frac{\partial J}{\partial b^{(1)}} = \frac{\partial J}{\partial H} \cdot \frac{\partial H}{\partial b^{(1)}} \qquad \frac{\partial H}{\partial b^{(1)}} = 1$$

$$= \delta^{(1)} = \begin{pmatrix} 0.0087 \\ 0.0099 \end{pmatrix}$$

使用梯度下降：

$$b^{(1)} = b^{(1)} - \alpha \frac{\partial J}{\partial b^{(1)}} = \begin{pmatrix} 0.35 \\ 0.35 \end{pmatrix} - 0.5 \cdot \begin{pmatrix} 0.0087 \\ 0.0099 \end{pmatrix}$$

$$= \begin{pmatrix} 0.3456 \\ 0.345 \end{pmatrix}$$

总结

以Ng课程为准，与上述实例有所不同（如 δ）

术语:

L：表示神经网络层数

$m^{(l)}$：第 l 层神经元个数

$f_{(l)}$：第 l 层神经元的激活值 ------ $g(z)$

$W^{(l)}$：表示 l 层到 $l+1$ 层的权重矩阵 ------ $\theta^{(l)}$

$b^{(l)}$：表示 l 层到 $l+1$ 层的偏置

$z^{(l)}$：表示 l 层神经元的净输入值

$a^{(l)}$：表示 l 层神经元的净输出值

$\delta^{(l)}$：表示 l 层的误差项

前馈神经网络

$$z^{(l)} = \theta^{(l-1)} \cdot a^{(l-1)} + b^{(l-1)}$$

$$a^{(l)} = f_l(z^{(l)})$$

式 $z^{(l)} = \theta^{(l-1)} \cdot f_{l-1}(z^{(l-1)}) + b^{(l-1)}$

$$a^{(l)} = f_l(\theta^{(l-1)} \cdot f_{l-1}(z^{(l-1)}) + b^{(l-1)})$$

反向传播算法

① 先计算净输入 $z^{(l)}$ 与激活值 $a^{(l)}$ ---- 前向✓

② 计算每一层的误差项 $\delta^{(l)}$.

1. 对于最后一层

$$\delta^{(L)} = \frac{dJ}{da} \odot f'_{(L)}(a)$$

2. 对于中间层

$$\delta^{(l)} = \left[(\theta^{(l)})^T \cdot \delta^{(l+1)}\right] \odot f'_l(z^{(l)})$$

其中 $f'_l(z^{(l)}) = \frac{da^{(l)}}{dz^{(l)}} = a^{(l)} \odot (1 - a^{(l)})$

3. 对于第一层

由于没有 $\delta^{(1)}$，因为第一层是输入层

③ 更新 $\theta^{(l)}$, $b^{(l)}$

$$\frac{\partial J}{\partial \theta^{(l)}} = \delta^{(l)} \cdot \left(a^{(l-1)}\right)^T \qquad \frac{\partial J}{\partial b^{(l)}} = \delta^{(l)}$$

④ 梯度下降

$$\theta^{(l)} = \theta^{(l)} - \alpha \cdot \frac{\partial J}{\partial \theta^{(l)}} = \theta^{(l)} - \alpha \cdot \left[\delta^{(l)} \cdot \left(a^{(l-1)}\right)\right]^T$$

$$b^{(l)} = b^{(l)} - \alpha \cdot \frac{\partial J}{\partial b^{(l)}} = b^{(l)} - \alpha \cdot \delta^{(l)}$$

9.5 梯度检验

参考视频: 9 - 5 - Gradient Checking (12 min).mkv

H3

当我们对一个较为复杂的模型（例如神经网络）使用梯度下降算法时，可能会存在一些不容易察觉的错误，意味着，虽然代价看上去在不断减小，但最终的结果可能并不是最优解。

为了避免这样的问题，我们采取一种叫做梯度检验**Numerical Gradient Checking**的数值检验方法。这种方法的思想是通过估计梯度值来检验我们计算的导数值是否真的是我们要求的。

1. 双侧差分**two-sided difference**

   在代价函数上沿着切线的方向选择离两个非常近的点，然后计算两个点的平均值用以估计梯度

   如估计在θ这一点的导数，对于这个特定的 θ，我们计算出在 θ-ε 处和 θ+ε 的代价值，然后求两个代价的平均，用以估计在 θ 处的代价值。

   ε是一个非常小的值，通常选取 0.001

$$\frac{\partial}{\partial\Theta}J(\Theta) \approx \frac{J(\Theta+\epsilon)-J(\Theta-\epsilon)}{2\epsilon}$$

2. 单侧差分**one-sided difference** ❎

$$\frac{\partial}{\partial\Theta}J(\Theta) \approx \frac{J(\Theta+\epsilon)-J(\Theta)}{\epsilon}$$

当θ是一个向量时，我们则需要对偏导数进行检验。因为代价函数的偏导数检验只针对一个参数的改变进行检验，

下面是一个只针对θ_1进行检验的示例：

$$\frac{\partial}{\partial\theta_1} = \frac{J\left(\theta_1+\varepsilon,\theta_2,\theta_3\ldots\theta_n\right)-J\left(\theta_1-\varepsilon,\theta_2,\theta_3\ldots\theta_n\right)}{2\varepsilon}$$

针对θ_n进行检验:

$$\frac{\partial}{\partial\theta_n} = \frac{J\left(\theta_1,\theta_2,\theta_3\ldots\theta_n+\varepsilon\right)-J\left(\theta_1,\theta_2,\theta_3\ldots\theta_n-\varepsilon\right)}{2\varepsilon}$$

根据上面的算法，计算出的偏导数存储在矩阵 $D_{ij}^{(l)}$ 中。检验时，我们要将该矩阵展开成为向量，同时我们也将 θ 矩阵展开为向量，我们针对每一个 θ 都计算一个近似的梯度值，将这些值存储于一个近似梯度矩阵中，最终将得出的这个矩阵同 $D_{ij}^{(l)}$ 进行比较。

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                    /(2*EPSILON);
end;
```

Check that `gradApprox` ≈ `DVec`

我们已经计算过deltaVector，而这个梯度检测的过程是非常耗费算力的，所以尽量在你认为有需要的位置做梯度监测，因此计算出gradApprox后就可以判断是否gradApprox ≈ deltaVector。一旦人为梯度没有问题，就将监测过程关闭，而不是在整个过程中，这个尤其要注意。

9.6 随机初始化

参考视频: 9 - 6 - Random Initialization (7 min).mkv

H3 任何优化算法都需要一些初始的参数。到目前为止我们都是初始所有参数为0，这样的初始方法对于逻辑回归来说是可行的，但是对于神经网络来说是不可行的。

如果我们令所有的初始参数都为0，这将意味着我们第二层的所有激活单元都会有相同的值。同理，如果我们初始所有的参数都为一个非0的数，结果也是一样的。

我们通常初始参数为$\Theta_{ij}^{(l)}$为$[-\epsilon,\epsilon]$之间的随机值，使用上述公式保证我们得到所需的界限，相同的程序使用于所有的Θ。

9.7 综合起来

参考视频: 9 - 7 - Putting It Together (14 min).mkv

小结一下使用神经网络时的步骤：

H3

1. 第一件要做的事是**选择网络结构**，即决定选择多少层以及决定每层分别有多少个单元。

   - 第一层的单元数即我们训练集的特征数量。
   - 最后一层的单元数是我们训练集的结果的类的数量。
     - y 需要用向量表示，如 $y = [1, 0, 0, \ldots, 0]^T$ ，而不能直接用某个数值表示
     - 如果隐藏层数大于1，确保每个隐藏层的单元个数相同。如上图三个隐藏层均有5个隐藏单元。
   - 通常情况下隐藏层单元的个数越多越好。
   - 我们真正要决定的是隐藏层的层数和每个中间层的单元数。
2. 训练神经网络：
   1. 参数的**随机初始化**

      Randomly initialize weights
   2. 利用**正向传播**方法计算所有的$h_\theta(x^{(i)})$

      Implement forward propagation to get $h_\theta(x^{(i)})$ for any $x^{(i)}$
   3. 编写计算**代价函数** $J(\Theta)$ 的代码

      Implement code to compute cost function $J(\Theta)$
   4. 利用**反向传播**方法计算所有偏导数$\frac{\partial}{\partial z_j^{(l)}} cost(i)$

      Implement backprop to compute partial derivatives $\frac{\partial}{\partial z_j^{(l)}} cost(i)$

具体参考 9.3节反向传播算法(2)

5. 利用**数值检验**方法检验这些偏导数

   Use gradient checking to compare $\frac{\partial}{\partial z_j^{(l)}} cost(i)$ computed using backpropagation vs using numerical estimate of gradient of $J(\Theta)$

   Then disable gradient checking code

6. 使用优化算法来**最小化代价函数**

   Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

9.8 自主驾驶

参考视频: 9 - 8 - Autonomous Driving (7 min).mkv

H3

在这段视频中，我想向你介绍一个具有历史意义的神经网络学习的重要例子。那就是使用神经网络来实现自动驾驶，也就是说使汽车通过学习来自己驾驶。接下来我将演示的这段视频是我从 Dean Pomerleau那里拿到的，他是我的同事，任职于美国东海岸的卡耐基梅隆大学。在这部分视频中，你就会明白可视化技术到底是什么？在看这段视频之前，我会告诉你可视化技术是什么。

在下面也就是左下方，就是汽车所看到的前方的路况图像。

在图中你依稀能看出一条道路，朝左延伸了一点，又向右了一点，然后上面的这幅图，你可以看到一条水平的菜单栏显示的是驾驶操作人选择的方向。就是这里的这条白亮的区段显示的就是人类驾驶者选择的方向。比如：最左边的区段，对应的操作就是向左急转，而最右端则对应向右急转的操作。因此，稍微靠左的区段，也就是中心稍微向左一点的位置，则表示在这一点上人类驾驶者的操作是慢慢的向左拐。

这幅图的第二部分对应的就是学习算法选出的行驶方向。并且，类似的，这一条白亮的区段显示的就是神经网络在这里选择的行驶方向，是稍微的左转，并且实际上在神经网络开始学习之前，你会看到网络的输出是一条灰色的区段，就像这样的一条灰色区段覆盖着整个区域这些均称的灰色区域，显示出神经网络已经随机初始化了，并且初始化时，我们并不知道汽车如何行驶，或者说我们并不知道所选行驶方向。只有在学习算法运行了足够长的时间之后，才会有这条白色的区段出现在整条灰色区域之中。显示出一个具体的行驶方向这就表

示神经网络算法，在这时候已经选出了一个明确的行驶方向，不像刚开始的时候，输出一段模糊的浅灰色区域，而是输出一条白亮的区段，表示已经选出了明确的行驶方向。

ALVINN (Autonomous Land Vehicle In a Neural Network)是一个基于神经网络的智能系统，通过观察人类的驾驶来学习驾驶，**ALVINN**能够控制**NavLab**，装在一辆改装版军用悍马，这辆悍马装载了传感器、计算机和驱动器用来进行自动驾驶的导航试验。实现**ALVINN**功能的第一步，是对它进行训练，也就是训练一个人驾驶汽车。

[Courtesy of Dean Pomerleau]

然后让**ALVINN**观看，**ALVINN**每两秒将前方的路况图生成一张数字化图片，并且记录驾驶者的驾驶方向，得到的训练集图片被压缩为30x32像素，并且作为输入提供给**ALVINN**的三层神经网络，通过使用反向传播学习算法，**ALVINN**会训练得到一个与人类驾驶员操纵方向基本相近的结果。一开始，我们的网络选择出的方向是随机的，大约经过两分钟的训练后，我们的神经网络便能够准确地模拟人类驾驶者的驾驶方向，对其他道路类型，也重复进行这个训练过程，当网络被训练完成后，操作者就可按下运行按钮，车辆便开始行驶了。

每秒钟**ALVINN**生成12次数字化图片，并且将图像传送给神经网络进行训练，多个神经网络同时工作，每一个网络都生成一个行驶方向，以及一个预测自信度的参数，预测自信度最高的那个神经网络得到的行驶方向。比如这里，在这条单行道上训练出的网络将被最终用于控制车辆方向，车辆前方突然出现了一个交叉十字路口，当车辆到达这个十字路口时，我们单行道网络对应的自信度骤减，当它穿过这个十字路口时，前方的双车道将进入其视线，双车道网络的自信度便开始上升，当它的自信度上升时，双车道的网络，将被选择来控制行驶方向，车辆将被安全地引导进入双车道路。

这就是基于神经网络的自动驾驶技术。当然，我们还有很多更加先进的试验来实现自动驾驶技术。在美国，欧洲等一些国家和地区，他们提供了一些比这个方法更加稳定的驾驶控制技术。但我认为，使用这样一个简单的基于反向传播的神经网络，训练出如此强大的自动驾驶汽车，的确是一次令人惊讶的成就。