神经网络基础

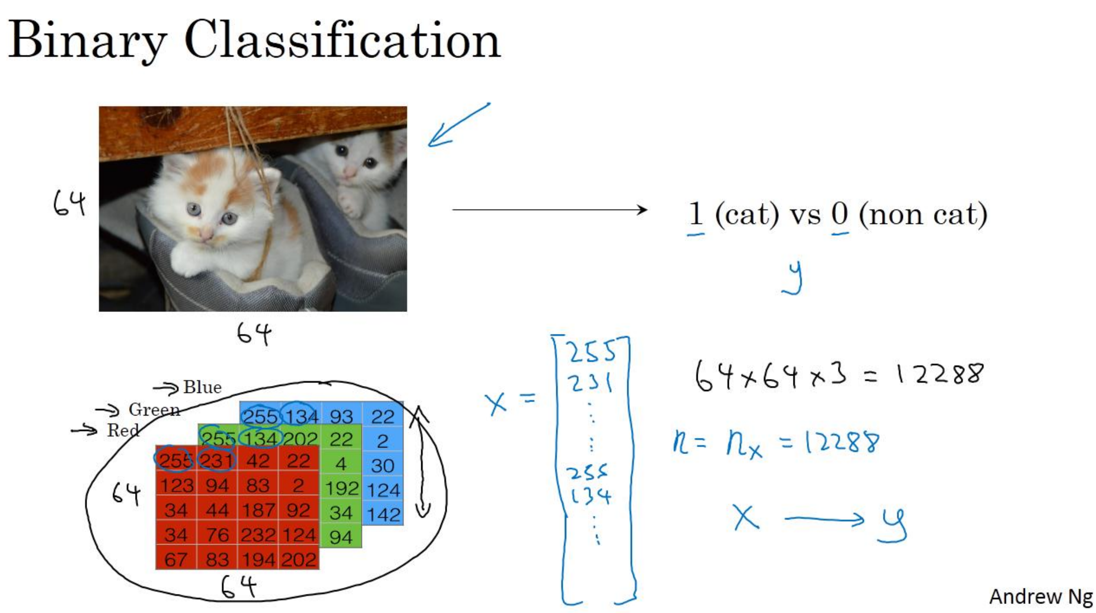
主要内容：

1、介绍二分类问题，以图片为例，将多维输入（x）转化为特征向量，输出（y）离散值 { 0, 1 }；
2、介绍逻辑回归（logistic）及其对应的代价函数（Cost function）形式；
3、介绍梯度下降算法，使用计算图（compution graph）描述神经网络的正向、反向传播过程；
4、在逻辑回归中使用梯度下降（Gradient Descent）算法，总结出优化权重参数 w 和偏置参数 b 的算法流程。

二分类（binary classification）

输出的 y 只有离散值 {0,1}（或者 {-1,1}）。

以一个图像识别问题为例，判断图片中是否有猫存在，0 代表无猫，1 代表有猫。



一般彩色图片包含 RGB 三个通道，首先将图片输入 x（维度是（64，64，3））转化为一维的特征向量（feature vector）。方法是每个通道逐行提取，最后连接起来。转化后的输入特征向量维度为

（64x64x3=12288，1）。此特征向量 x 是列向量，维度一般记为 nx。

逻辑回归(logistic regression)

逻辑回归中，预测值$\hat{h} = P(y=1|x)$表示为 1 的概率，取值范围在[0,1]之间。

使用线性模型引入权重参数 w 和偏置参数 b。w 的维度是$(n_x,1)$，b 是一个常数项。则逻辑回归的线性预测为：

$$\hat{y} = w^T x + b$$

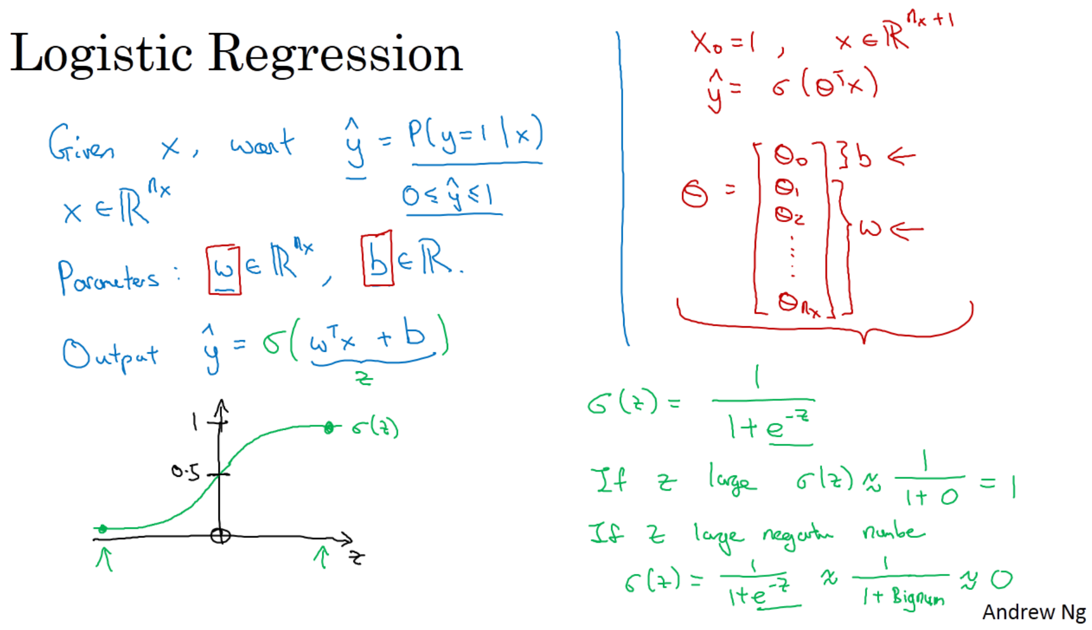
上式的输出范围是整个实数范围，为了使 y 属于[0,1]，引入 sigmoid 函数，让输出限定在[0,1]之间。则：

$$\hat{y} = sigmoid(w^T x + b) = \sigma(w^T x + b)$$

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

Sigmoid 一阶导数可用其自身表示：

$$\sigma^{'}(z) = \sigma(z)(1-\sigma(z))$$



代价函数(cost function)

通过优化代价函数，得到对应的 w 和 b。从单个样本的角度，希望样本的预测值$\hat{y}$和真实值y越接近越好，若使用平方误差（squared error），如下：

$$L(\hat{y},y)=\frac{1}{2}(\hat{y}-y)^2$$

但逻辑回归，一般不用上式作为损失函数，因为上式是非凸函数，非凸函数在梯度下降时，容易得到局部最小值，所以一般选的 LOSS function 是凸函数。

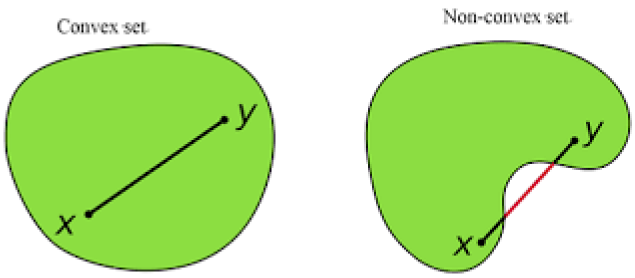


图 1 凸函数与非凸函数

所以选用如下 Loss function:

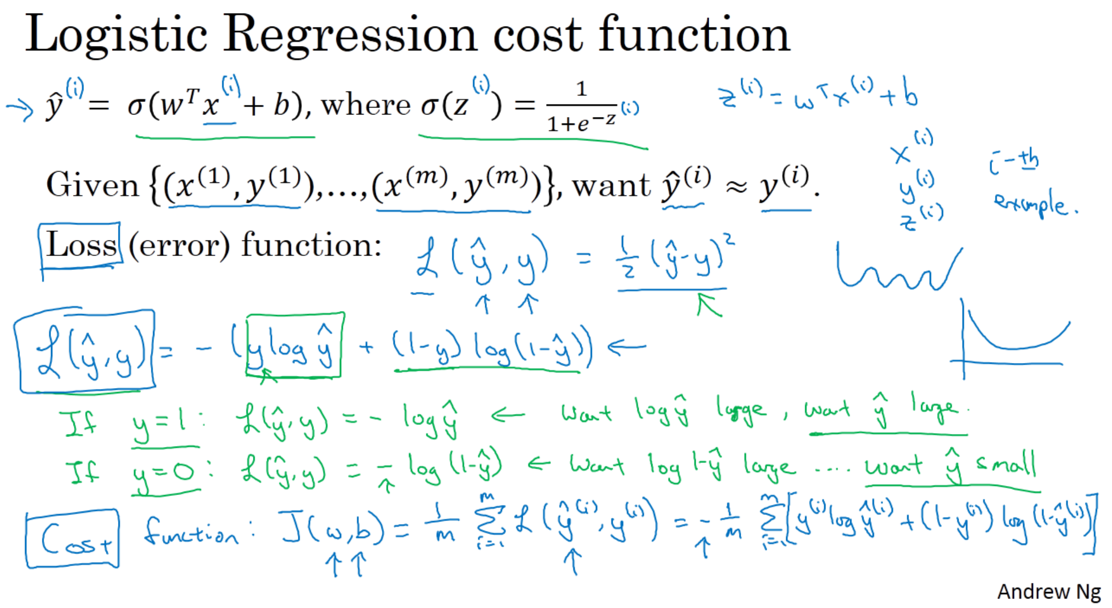
$$L(\hat{y},y)=-(y\log\hat{y}+(1-y)\log(1-\hat{y}))$$

当y=1时， $L(\hat{y},y)=-log\,\hat{y}$ 。如果 $\hat{y}$ 越接近1， $L(\hat{y},y)\approx 0$ ，表示预测效果越好；如果 $\hat{y}$ 越接近0， $L(\hat{y},y)\approx +\infty$ ，表示预测效果越差。这正是我们希望Loss function所实现的功能。

当y=0时， $L(\hat{y},y)=-log\,(1-\hat{y})$ 。如果 $\hat{y}$ 越接近0， $L(\hat{y},y)\approx 0$ ，表示预测效果越好；如果 $\hat{y}$ 越接近1， $L(\hat{y},y)\approx +\infty$ ，表示预测效果越差。这也正是我们希望Loss function所实现的功能。

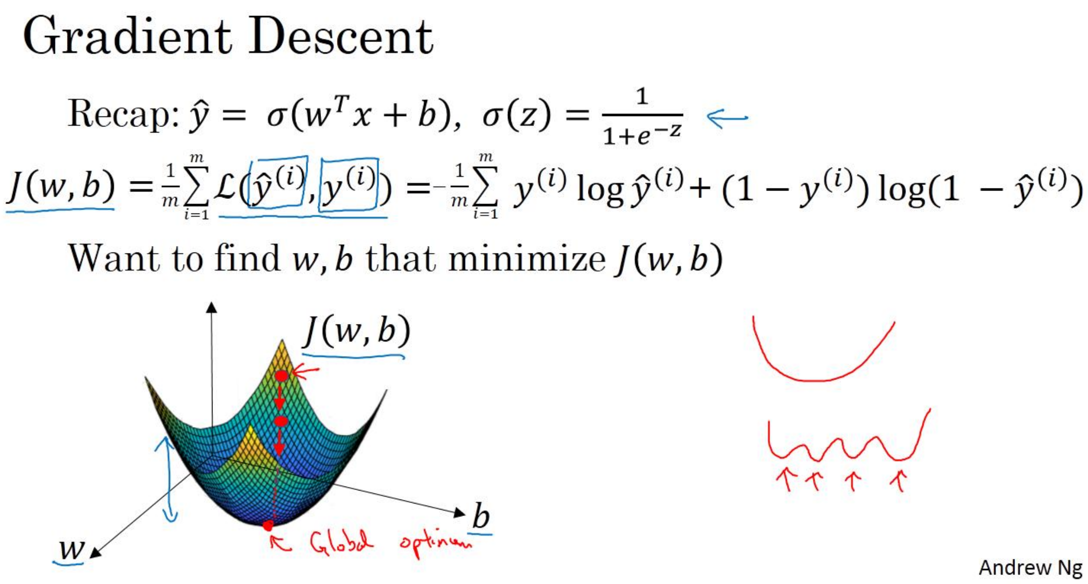
上面是单个样本，m 个样本的 Cost function 反应的是 m 个样本的平均接近程度。

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L(\hat{y}^{(i)},y^{(i)})=-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log\hat{y}^{(i)}+(1-y^{(i)})\log(1-\hat{y}^{(i)}))$$



梯度下降（gradient descent）

由于 J(w,b)是凸函数，梯度下降算法是先随机选择一组参数 w 和 b，然后迭代的过程中分别沿着 w 和 b 的梯度的反方向前进一小步，不断修正 w 和 b。



梯度下降算法每次迭代更新，w 和 b 的更新表达式为：

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

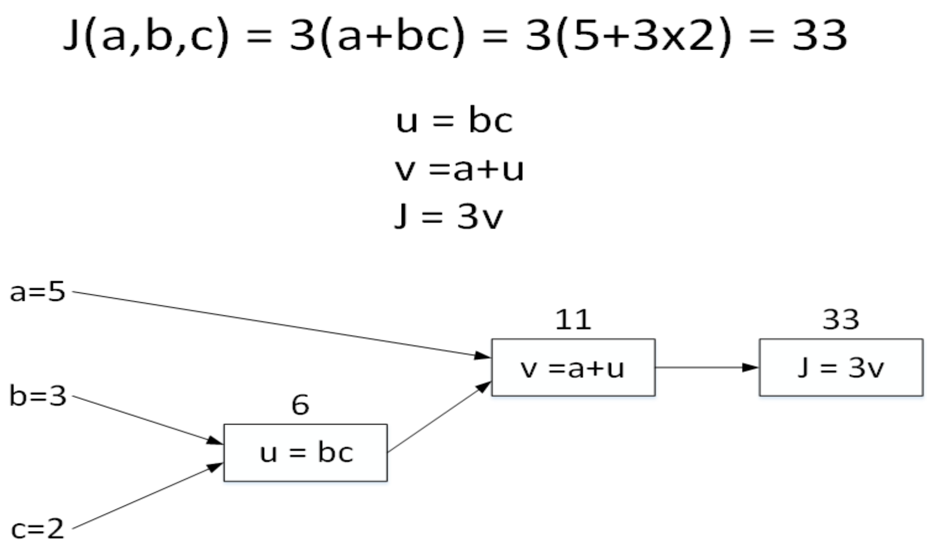
$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

上式中 α 是学习因子（learning rate）

计算图(Computation graph)

整个神经网络的训练包含两个过程：正向传播(Forward Propagation)和反向传播。正向传播是从输入到输出，由神经网络计算得到预测输出的过程；反向传播是从输出到输入，对参数 w 和 b 计算梯度的过程。下面，我们用计算图（Computation graph）的形式来理解这两个过程。

举个简单的例子，假如 Cost function 为 J(a,b,c)=3(a+bc)，包含 a，b，c 三个变量。我们用 u 表示 bc，v 表示 a+u，则 J=3v。它的计算图可以写成如下图所示：



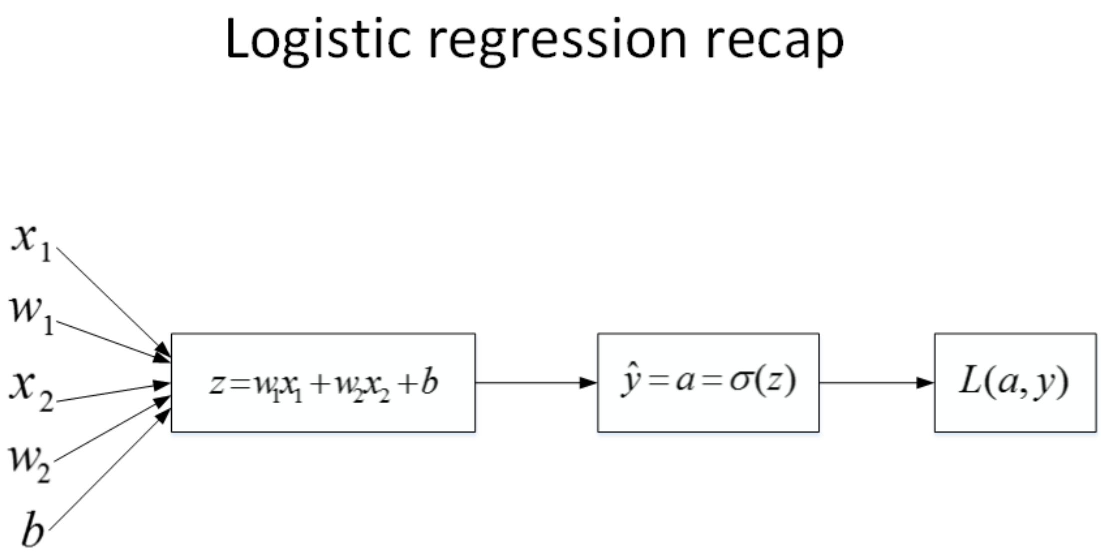
逻辑回归的梯度下降

对单个样本而言，逻辑回归 Loss function 表达式如下：

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$L(a, y) = -(y\log(a) + (1-y)\log(1-a))$$

根据上述公式，例如输入样本x有两个特征(x_1, x_2)，相应的权重w维度也是2，即(w_1, w_2)。则$z = w_1 x_1 + w_2 x_2 + b$

正向传播过程如下图



反向传播过程即由 Loss function 计算参数 w 和 b 的偏导数。推导过程如下：

$$da = \frac{\partial L}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = (-\frac{y}{a} + \frac{1-y}{1-a}) \cdot a(1-a) = a - y$$

知道了dz之后，就可以直接对 w_1 ， w_2 和b进行求导了。

$$dw_1 = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = x_1 \cdot dz = x_1(a-y)$$

$$dw_2 = \partial^2 \partial L \partial w_2 = \partial^2 \partial L \partial z \cdot \partial^2 \partial z \partial w_2 = x_2 \cdot dz = x_2(a-y)$$

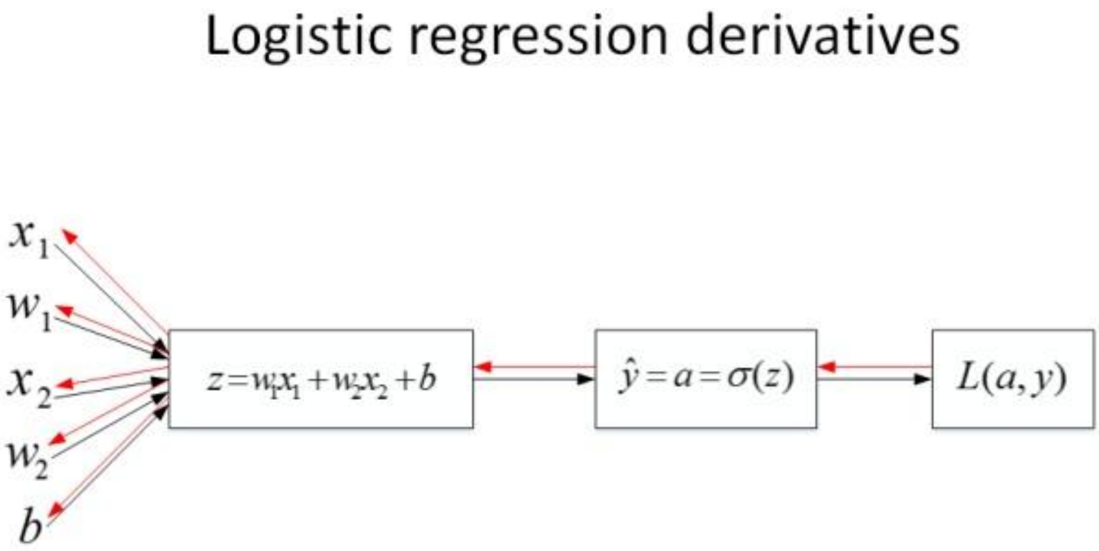
$$db = \partial^2 \partial L \partial b = \partial^2 \partial L \partial z \cdot \partial^2 \partial z \partial b = 1 \cdot dz = a - y$$

则梯度下降算法可表示为：

w1:=w1−α dw1

w2:=w2−α dw2

b:=b−α db



M 个样本的梯度下降

上一部分讲的是对单个样本求偏导和梯度下降。如果有 m 个样本，其 Cost function 表达式如下：

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \sigma(z^{(i)})$$

$$J(w,b) = \partial^2 1m \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\partial^2 1m \sum_{i=1}^{m} [y^{(i)} log\, \hat{y}^{(i)} + (1 - y^{(i)}) log\, (1 - \hat{y}^{(i)})]$$

Cost function 关于 w 和 b 的偏导数可以写成和平均的形式：

$$dw_1 = \frac{1}{m}\sum_{i=1}^{m} x_1^{(i)}(a^{(i)} - y^{(i)})$$

$$dw_2 = \frac{1}{m}\sum_{i=1}^{m} x_2^{(i)}(a^{(i)} - y^{(i)})$$

$$db = \partial^2 1m\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

这样，每次迭代中 w 和 b 的梯度有 m 个训练样本计算平均值得到。其算法流程图如下所示：

```
J=0; dw1=0; dw2=0; db=0;
for i = 1 to m
       z(i) = wx(i)+b;
       a(i) = sigmoid(z(i));
       J += -[y(i)log(a(i))+(1-y(i)）log(1-a(i));
       dz(i) = a(i)-y(i);
       dw1 += x1(i)dz(i);
       dw2 += x2(i)dz(i);
       db += dz(i);
J /= m;
dw1 /= m;
dw2 /= m;
db /= m;
```

经过每次迭代后，根据梯度下降算法，w和b都进行更新：

$$w_1 := w_1 - \alpha\partial^2 dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

$$b := b - \alpha\, db$$

这样经过n次迭代后，整个梯度下降算法就完成了。