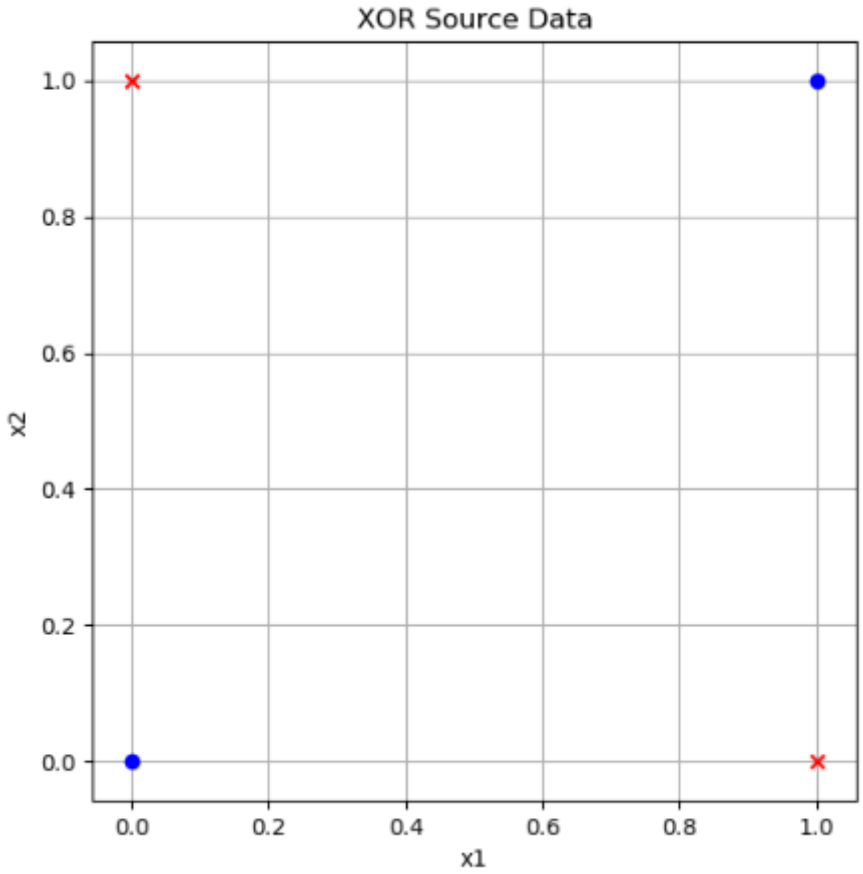
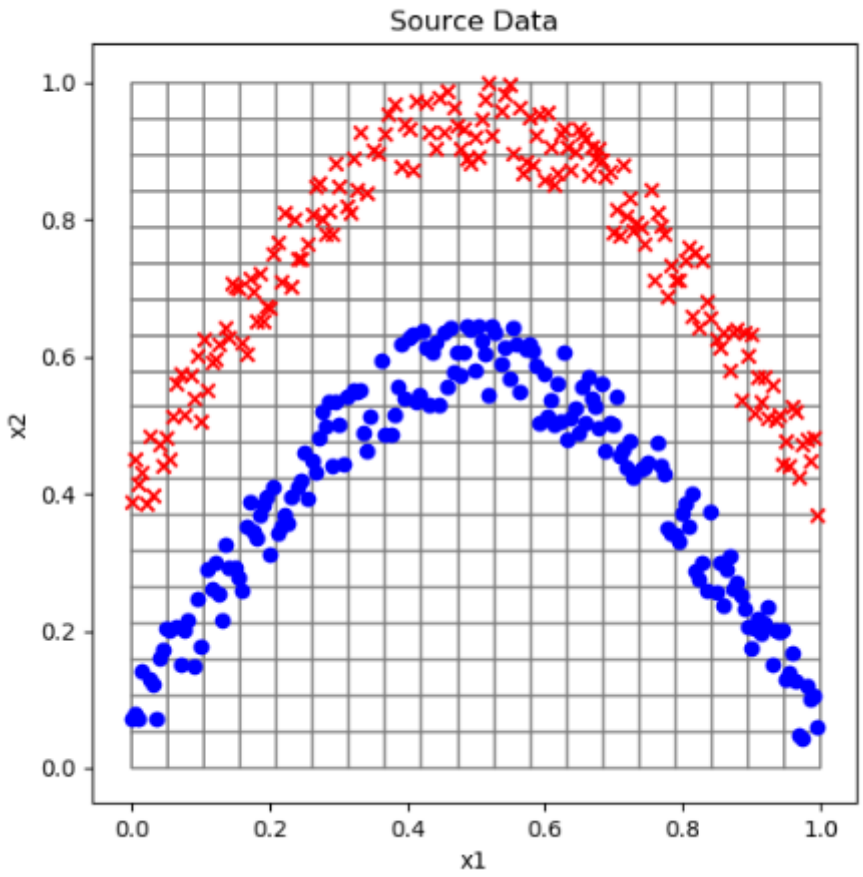
1. 非线性分类

1.1 多入单出的双层神经网络 - 非线性二分类

1.1.1 非线性二分类问题

异或问题 + 双弧形问题





1.1.2 二分类模型的评估标准

1.1.2.1 准确率 Accuracy

对于二分类问题，假设测试集上一共1000个样本，其中550个正例，450个负例。测试一个模型时，得到的结果是：521个正例样本被判断为正类，435个负例样本被判断为负类，则正确率计算如下：

$$Accuracy = (521 + 435)/1000 = 0.956$$

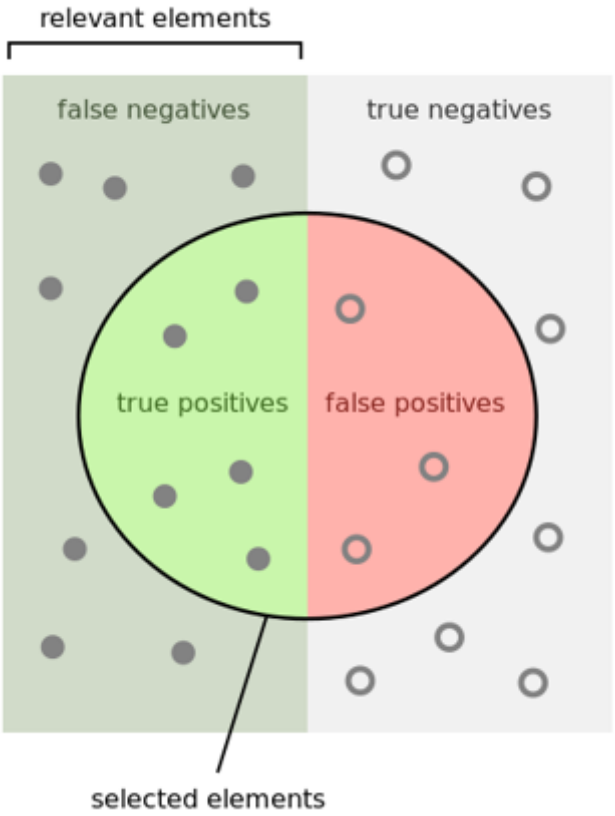
但是这种计算方法丢失了很多细节，比如：是正类判断的精度高还是负类判断的精度高呢？因此，下一种评估标准。

1.1.2.2 混淆矩阵

还是用上面的例子，如果具体深入到每个类别上，会分成4部分来评估：

- 正例中被判断为正类的样本数（TP-True Positive）：521
- 正例中被判断为负类的样本数（FN-False Negative）：550-521=29
- 负例中被判断为负类的样本数（TN-True Negative）：435
- 负例中被判断为正类的样本数（FP-False Positive）：450-435=15

可以用下图来帮助理解。



- 左侧实心圆点是正类，右侧空心圆是负类；
- 在圆圈中的样本是被模型判断为正类的，圆圈之外的样本是被判断为负类的；
- 左侧圆圈外的点是正类但是误判为负类，右侧圆圈内的点是负类但是误判为正类；
- 左侧圆圈内的点是正类且被正确判别为正类，右侧圆圈外的点是负类且被正确判别为负类。

预测值	**被判断为正类**	**被判断为负类**	**Total**
样本实际为正例	TP-True Positive	FN-False Negative	Actual Positive=TP+FN
样本实际为负例	FP-False Positive	TN-True Negative	Actual Negative=FP+TN
Total	Predicated Postivie=TP+FP	Predicated Negative=FN+TN	

从混淆矩阵中可以得出以下统计指标：

- 准确率 Accuracy

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

$$= \frac{521+435}{521+29+435+15} = 0.956$$

这个指标就是上面提到的准确率，越大越好。

- 精确率/查准率 Precision

分子为被判断为正类并且真的是正类的样本数，分母是被判断为正类的样本数。越大越好。

$$Precision = \frac{TP}{TP+FP} = \frac{521}{521+15} = 0.972$$

- 召回率/查全率 Recall

$$Recall = \frac{TP}{TP + FN} = \frac{521}{521 + 29} = 0.947$$

分子为被判断为正类并且真的是正类的样本数，分母是真的正类的样本数。越大越好。

- TPR - True Positive Rate 真正例率

$$TPR = \frac{TP}{TP + FN} = Recall = 0.947$$

- FPR - False Positive Rate 假正例率

$$FPR = \frac{FP}{FP + TN} = \frac{15}{15 + 435} = 0.033$$

分子为被判断为正类的负例样本数，分母为所有负类样本数。越小越好。

- 调和平均值 F1

$$F1 = \frac{2 \times Precision \times Recall}{recision + Recall}$$

$$= \frac{2 \times 0.972 \times 0.947}{0.972 + 0.947} = 0.959$$

该值越大越好。

- ROC曲线与AUC

ROC，Receiver Operating Characteristic，接收者操作特征，又称为感受曲线（Sensitivity Curve），是反映敏感性和特异性连续变量的综合指标，曲线上各点反映着相同的感受性，它们都是对同一信号刺激的感受性。
ROC曲线的横坐标是FPR，纵坐标是TPR。

AUC，Area Under Roc，即ROC曲线下面的面积。

在二分类器中，如果使用Logistic函数作为分类函数，可以设置一系列不同的阈值，比如[0.1,0.2,0.3...0.9]，把测试样本输入，从而得到一系列的TP、FP、TN、FN，然后就可以绘制如下曲线，如下图：

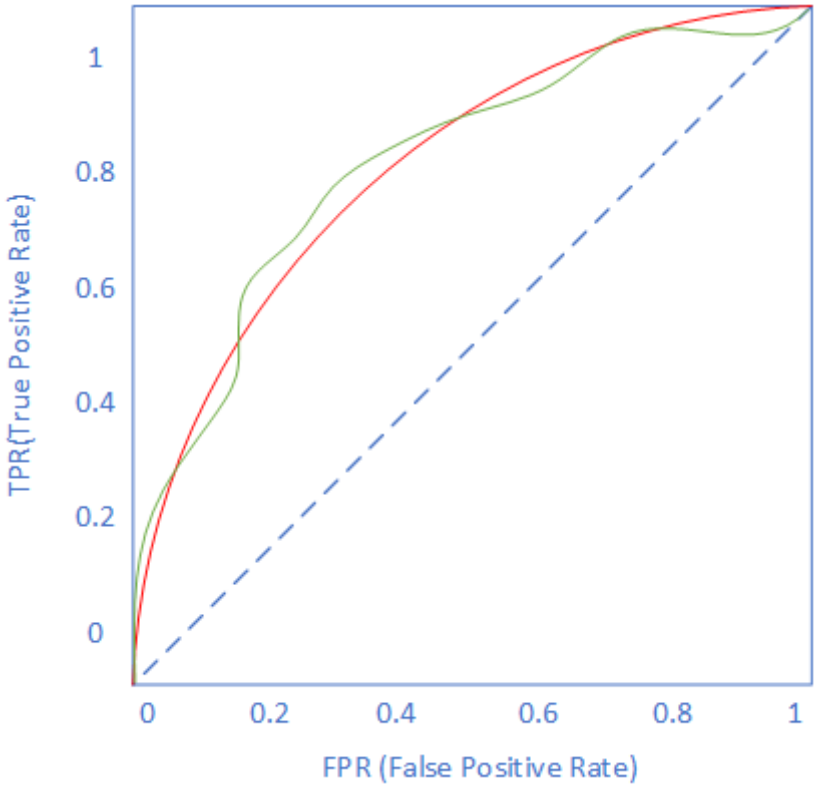


图10-4 ROC曲线图

图中红色的曲线就是ROC曲线，曲线下的面积就是AUC值，取值区间为$[0.5, 1.0]$，面积越大越好。

- ROC曲线越靠近左上角，该分类器的性能越好。
- 对角线表示一个随机猜测分类器。
- 若一个学习器的ROC曲线被另一个学习器的曲线完全包住，则可判断后者性能优于前者。
- 若两个学习器的ROC曲线没有包含关系，则可以判断ROC曲线下的面积，即AUC，谁大谁好。

既然已经这么多标准，为什么还要使用ROC和AUC呢？**因为ROC曲线有个很好的特性：当测试集中的正负样本的分布变换的时候，ROC曲线能够保持不变。**

1.1.2.3 Mean absolute error 和 Root mean squared error

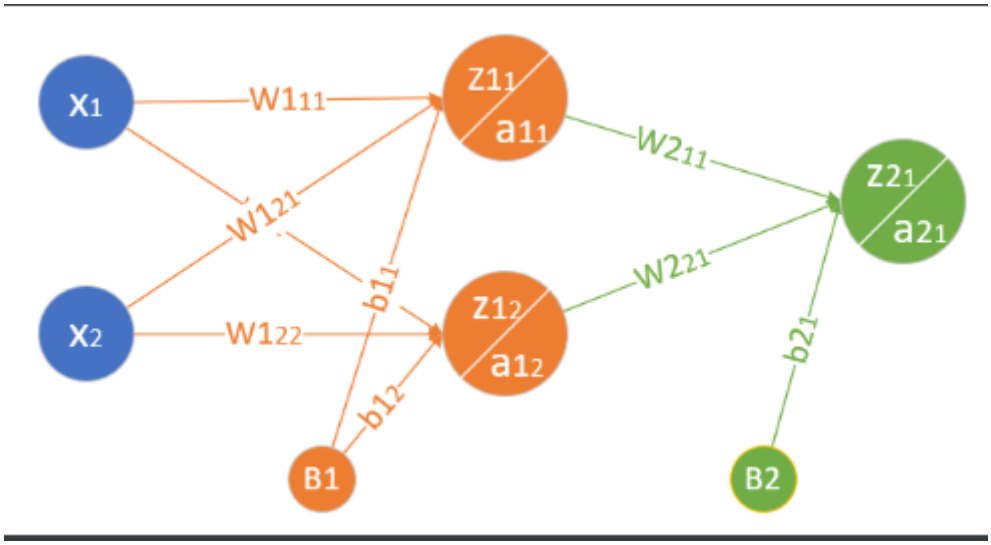
平均绝对误差和均方根误差，用来衡量分类器预测值和实际结果的差异，越小越好。

1.2.2.4 Relative absolute error 和 Root relative squared error

相对绝对误差和相对均方根误差，有时绝对误差不能体现误差的真实大小，而相对误差通过体现误差占真值的比重来反映误差大小。

1.1.3 非线性二分类实现

1.1.3.1 定义神经网络结构



- 输入层两个特征值x_1, x_2

$$X = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$

- 隐层2×2的权重矩阵$W1$

$$W1 = \begin{pmatrix} w1_{11} & w1_{12} \\ w1_{21} & w1_{22} \end{pmatrix}$$

- 隐层1×2的偏移矩阵$B1$

$$B1 = \begin{pmatrix} b1_1 & b1_2 \end{pmatrix}$$

- 隐层由两个神经元构成

$$Z1 = \begin{pmatrix} z1_1 & z1_2 \end{pmatrix}$$

$$A1 = \begin{pmatrix} a1_1 & a1_2 \end{pmatrix}$$

- 输出层2×1的权重矩阵$W2$

$$W2 = \begin{pmatrix} w2_{11} \\ w2_{21} \end{pmatrix}$$

- 输出层1×1的偏移矩阵$B2$

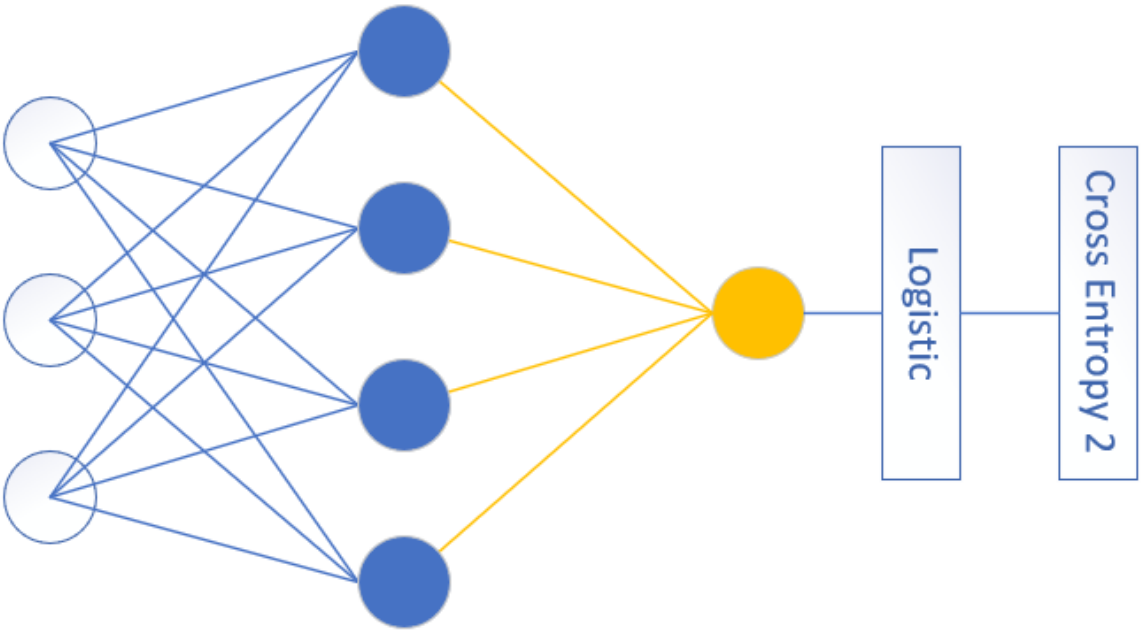
$$B2 = \begin{pmatrix} b2_1 \end{pmatrix}$$

- 输出层有一个神经元使用Logistic函数进行分类

$$Z2 = \begin{pmatrix} z2_1 \end{pmatrix}$$

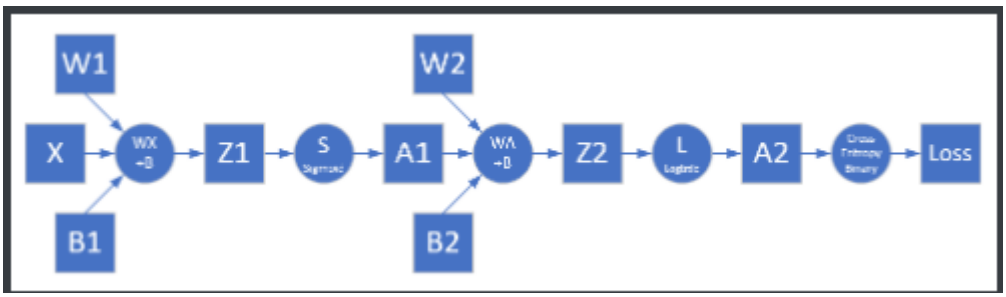
$$A2 = \begin{pmatrix} a2_1 \end{pmatrix}$$

对于一般的用于二分类的双层神经网络可以是下图的样子。



输入特征值可以有很多，隐层单元也可以有很多，输出单元只有一个，且后面要接Logistic分类函数和二分类交叉熵损失函数。

1.1.3.2 向前计算



第一层

- 线性计算

$$z1_1 = x_1 w1_{11} + x_2 w1_{21} + b1_1$$

$$z1_2 = x_1 w1_{12} + x_2 w1_{22} + b1_2$$

$$Z1 = X \cdot W1 + B1$$

- 激活函数

$$a1_1 = Sigmoid(z1_1)$$

$$a1_2 = Sigmoid(z1_2)$$

$$A1 = (a1_1 \quad a1_2) = Sigmoid(Z1)$$

第二层

- 线性计算

$$z2_1 = a1_1 w2_{11} + a1_2 w2_{21} + b2_1$$

$$Z2 = A1 \cdot W2 + B2$$

- 分类函数

$$a2_1 = Logistic(z2_1)$$
$$A2 = Logistic(Z2)$$

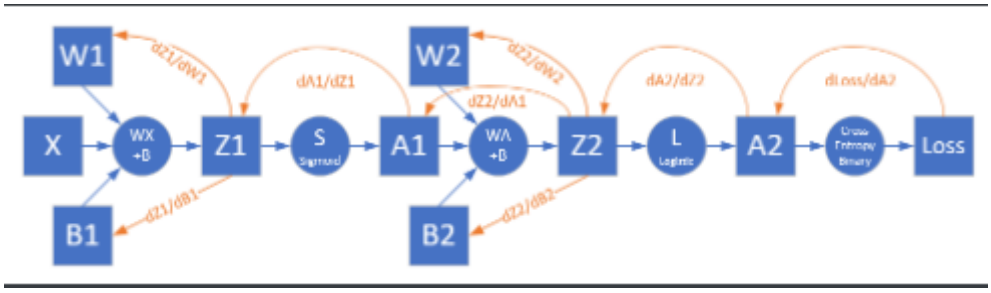
损失函数

我们把异或问题归类成二分类问题，所以使用二分类交叉熵损失函数：

$$loss = -Y \ln A2 + (1-Y)\ln(1-A2) \qquad (12)$$

在二分类问题中，$Y, A2$都是一个单一的数值，而非矩阵，但是为了前后统一，我们可以把它们看作是一个1×1的矩阵。

1.1.3.3 反向传播



1.1.4 实现逻辑异或门

思考一下，神经网络最后是通过什么方式判定样本的类别呢？在前向计算过程中，最后一个公式是Logistic函数，它把$(-\infty, +\infty)$压缩到了(0,1)之间，相当于计算了一个概率值，然后通过概率值大于0.5与否，判断是否属于正类。虽然异或问题只有4个样本点，但是如果：

1. 我们在[0,1]正方形区间内进行网格状均匀采样，这样每个点都会有坐标值；
2. 再把坐标值代入神经网络进行推理，得出来的应该会是一个网格状的结果；
3. 每个结果都是一个概率值，肯定处于(0,1)之间，所以不是大于0.5，就是小于0.5；
4. 我们把大于0.5的网格涂成粉色，把小于0.5的网格涂成黄色，就应该可以画出分界线来了。

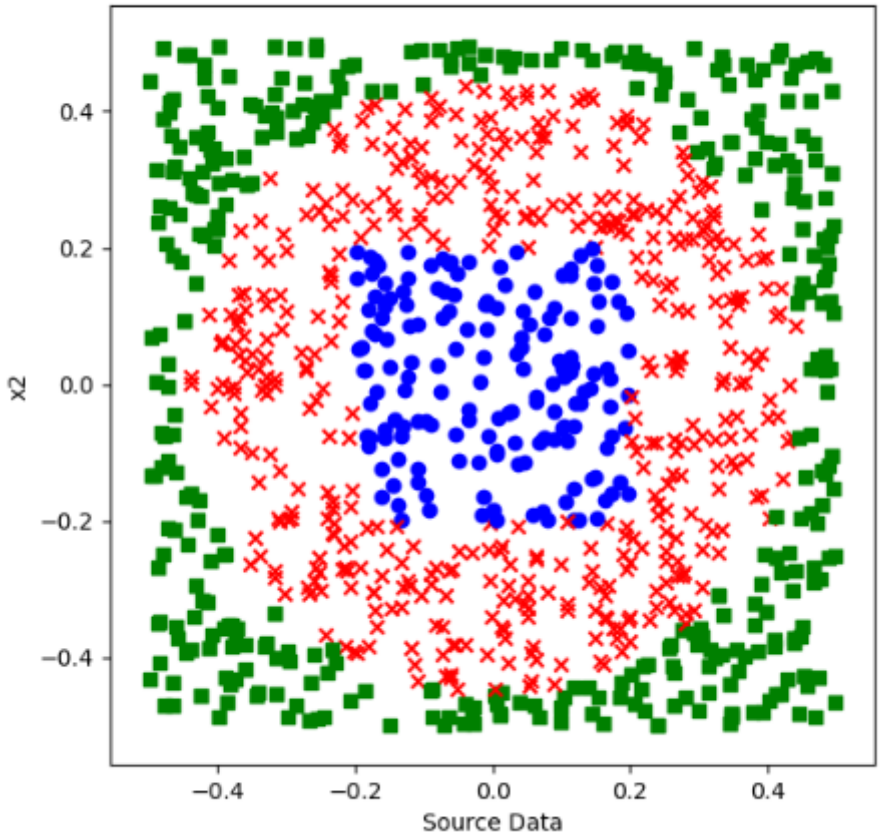
1.1.5 实现双弧形二分类

神经网络通过空间变换的方式，把线性不可分的样本变成了线性可分的样本，从而给最后的分类变得很容易。

1.2 多入多出的双层神经网络 - 非线性多分类

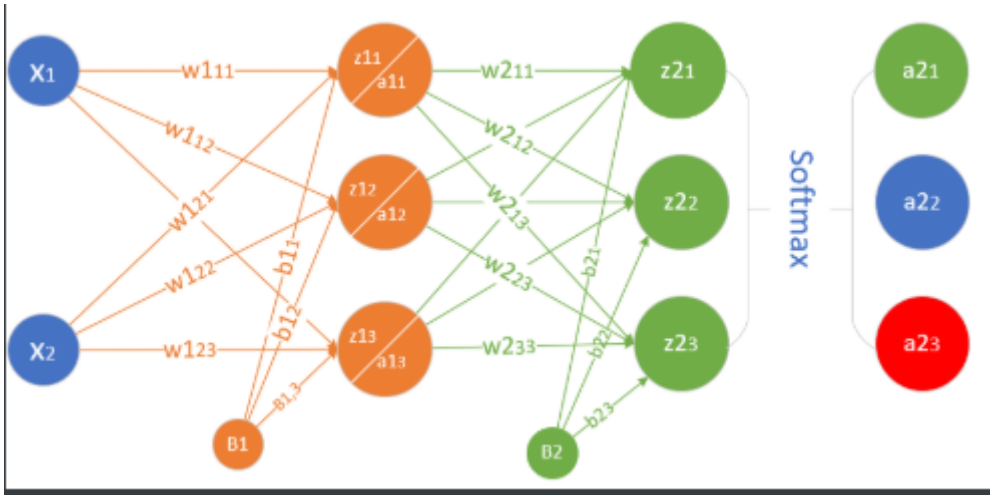
1.2.1 非线性二分类问题

铜钱孔形问题



1.2.2 非线性多分类实现

1.2.1.1 定义神经网络结构



- 输入层两个特征值x_1, x_2

$$x = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$$

- 隐层2×3的权重矩阵$W1$

$$W1 = \begin{pmatrix} w1_{11} & w1_{12} & w1_{13} \\ w1_{21} & w1_{22} & w1_{23} \end{pmatrix}$$

- 隐层1×3的偏移矩阵$B1$

$$B1 = \begin{pmatrix} b1_1 & b1_2 & b1_3 \end{pmatrix}$$

- 隐层由3个神经元构成
- 输出层3×3的权重矩阵$W2$

$$W2 = \begin{pmatrix} w2_{11} & w2_{12} & w2_{13} \\ w2_{21} & w2_{22} & w2_{23} \\ w2_{31} & w2_{32} & w2_{33} \end{pmatrix}$$

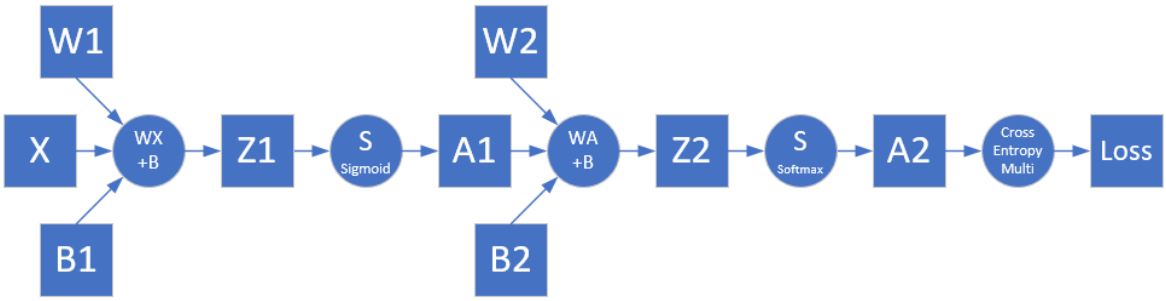
- 输出层1×1的偏移矩阵$B2$

$$B2 = \begin{pmatrix} b2_1 & b2_2 & b2_3 \end{pmatrix}$$

- 输出层有3个神经元使用Softmax函数进行分类

1.2.1.2 前向计算

根据网络结构，可以绘制前向计算图，如图所示。



第一层

- 线性计算

$$z1_1 = x_1 w1_{11} + x_2 w1_{21} + b1_1$$

$$z1_2 = x_1 w1_{12} + x_2 w1_{22} + b1_2$$

$$z1_3 = x_1 w1_{13} + x_2 w1_{23} + b1_3$$

$$Z1 = X \cdot W1 + B1$$

- 激活函数

$$a1_1 = Sigmoid(z1_1)$$

$$a1_2 = Sigmoid(z1_2)$$

$$a1_3 = Sigmoid(z1_3)$$

$$A1 = Sigmoid(Z1)$$

第二层

- 线性计算

$$z2_1 = a1_1 w2_{11} + a1_2 w2_{21} + a1_3 w2_{31} + b2_1$$

$$z2_2 = a1_1 w2_{12} + a1_2 w2_{22} + a1_3 w2_{32} + b2_2$$

$$z2_3 = a1_1 w2_{13} + a1_2 w2_{23} + a1_3 w2_{33} + b2_3$$

$$Z2 = A1 \cdot W2 + B2$$

- 分类函数

$$a2_1 = \frac{e^{z2_1}}{e^{z2_1} + e^{z2_2} + e^{z2_3}}$$

$$a2_2 = \frac{e^{z2_2}}{e^{z2_1} + e^{z2_2} + e^{z2_3}}$$

$$a2_3 = \frac{e^{z2_3}}{e^{z2_1} + e^{z2_2} + e^{z2_3}}$$

$$A2 = Softmax(Z2)$$

损失函数

使用多分类交叉熵损失函数：

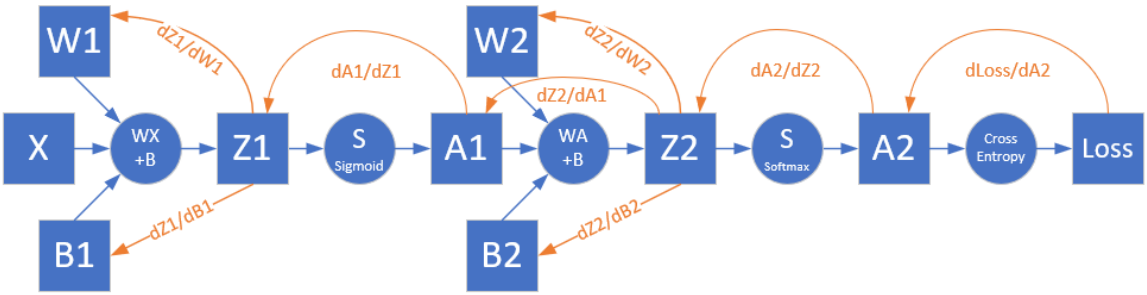
$$loss = -(y_1 \ln a2_1 + y_2 \ln a2_2 + y_3 \ln a2_3)$$

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij} \ln(a2_{ij})$$

m为样本数，n为类别数。

1.2.1.3 反向传播

根据前向计算图，可以绘制出反向传播的路径如下图：



1.3 分类样本不平衡问题

以二分类为例，比如正负例都各有1000个左右。如果是1200:800的比例，也是可以接受的，但是如果是1900:100，就需要有些措施来解决不平衡问题了，否则最后的训练结果很大可能是忽略了负例，将所有样本都分类为正类了。

如果是三分类，假设三个类别的样本比例为：1000:800:600，这是可以接受的；但如果是1000:300:100，就属于不平衡了。它带来的结果是分类器对第一类样本过拟合，而对其它两个类别的样本欠拟合，测试效果一定很糟糕。

1.3.1 如何解决样本不平衡问题

1.3.1.1 平衡数据集

一些经验法则：

- 考虑对大类下的样本（超过1万、十万甚至更多）进行欠采样，即删除部分样本；
- 考虑对小类下的样本（不足1万甚至更少）进行过采样，即添加部分样本的副本；
- 考虑尝试随机采样与非随机采样两种采样方法；

- 考虑对各类别尝试不同的采样比例，比一定是1:1，有时候1:1反而不好，因为与现实情况相差甚远；
- 考虑同时使用过采样（over-sampling）与欠采样（under-sampling）。

1.3.1.2 尝试其它评价指标

常规的分类评价指标可能会失效，比如将所有的样本都分类成大类，那么准确率、精确率等都会很高。这种情况下，AUC是最好的评价指标。

1.3.1.3 尝试产生人工数据样本

一种简单的人工样本数据产生的方法便是，对该类下的所有样本每个属性特征的取值空间中随机选取一个组成新的样本，即属性值随机采样。

可以使用基于经验对属性值进行随机采样而构造新的人工样本，或者使用类似朴素贝叶斯方法假设各属性之间互相独立进行采样，这样便可得到更多的数据，但是无法保证属性之前的线性关系（如果本身是存在的）。

1.3.1.4 集成学习

一个很好的方法去处理非平衡数据问题，并且在理论上证明了。这个方法便是由Robert E. Schapire于1990年在Machine Learning提出的"The strength of weak learnability"，该方法是一个boosting算法，它递归地训练三个弱学习器，然后将这三个弱学习器结合起形成一个强的学习器。我们可以使用这个算法的第一步去解决数据不平衡问题。

1. 首先使用原始数据集训练第一个学习器L1；
2. 然后使用50%在L1学习正确和50%学习错误的那些样本训练得到学习器L2，即从L1中学习错误的样本集与学习正确的样本集中，循环一边采样一个；
3. 接着，使用L1与L2不一致的那些样本去训练得到学习器L3；
4. 最后，使用投票方式作为最后输出。

那么如何使用该算法来解决类别不平衡问题呢?

假设是一个二分类问题，大部分的样本都是true类。让L1输出始终为true。使用50%在L1分类正确的与50%分类错误的样本训练得到L2，即从L1中学习错误的样本集与学习正确的样本集中，循环一边采样一个。因此，L2的训练样本是平衡的。L使用L1与L2分类不一致的那些样本训练得到L3，即在L2中分类为false的那些样本。最后，结合这三个分类器，采用投票的方式来决定分类结果，因此只有当L2与L3都分类为false时，最终结果才为false，否则true。

1.4 多入多出的三层神经网络 - 深度非线性多分类

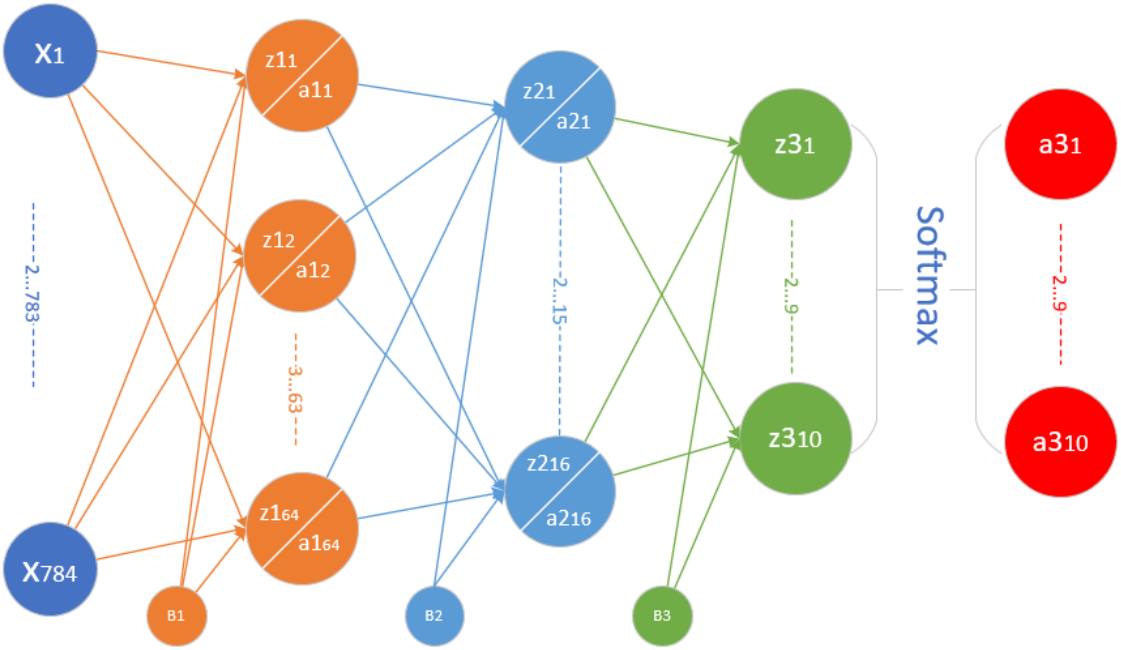
1.4.1 多变量非线性多分类问题

MNIST数字手写体识别图片集

1.4.2 三层神经网络的实现

1.4.2.1 定义神经网络

为了完成MNIST分类，设计一个三层神经网络结构



输入层

共计$28 \times 28 = 784$个特征值：

$$X = \begin{pmatrix} x_1 & x_2 & \cdots & x_{784} \end{pmatrix}$$

隐层1

- 权重矩阵$W1$形状为784×64

$$W1 = \begin{pmatrix} w1_{1,1} & w1_{1,2} & \cdots & w1_{1,64} \\ \vdots & \vdots & \cdots & \vdots \\ w1_{784,1} & w1_{784,2} & \cdots & w1_{784,64} \end{pmatrix}$$

- 偏移矩阵$B1$的形状为1×64

$$B1 = \begin{pmatrix} b1_1 & b1_2 & \cdots & b1_{64} \end{pmatrix}$$

- 隐层1由64个神经元构成，其结果为1×64的矩阵

$$Z1 = \begin{pmatrix} z1_1 & z1_2 & \cdots & z1_{64} \end{pmatrix}$$

$$A1 = \begin{pmatrix} a1_1 & a1_2 & \cdots & a1_{64} \end{pmatrix}$$

隐层2

- 权重矩阵$w2$形状为64×16

$$W2 = \begin{pmatrix} w2_{1,1} & w2_{1,2} & \cdots & w2_{1,16} \\ \vdots & \vdots & \cdots & \vdots \\ w2_{64,1} & w2_{64,2} & \cdots & w2_{64,16} \end{pmatrix}$$

- 偏移矩阵#B2#的形状是1×16

$$B2 = \begin{pmatrix} b2_1 & b2_2 & \cdots & b2_{16} \end{pmatrix}$$

- 隐层2由16个神经元构成

$$Z2 = \begin{pmatrix} z2_1 & z2_2 & \cdots & z2_{16} \end{pmatrix}$$

$$A2 = \begin{pmatrix} a2_1 & a2_2 & \cdots & a2_{16} \end{pmatrix}$$

输出层

- 权重矩阵$W3$的形状为16×10

$$W3 = \begin{pmatrix} w3_{1,1} & w3_{1,2} & \cdots & w3_{1,10} \\ \vdots & \vdots & \cdots & \vdots \\ w3_{16,1} & w3_{16,2} & \cdots & w3_{16,10} \end{pmatrix}$$

- 输出层的偏移矩阵$B3$的形状是1×10

$$B3 = \begin{pmatrix} b3_1 & b3_2 & \cdots & b3_{10} \end{pmatrix}$$

- 输出层有10个神经元使用Softmax函数进行分类

$$Z3 = \begin{pmatrix} z3_1 & z3_2 & \cdots & z3_{10} \end{pmatrix}$$

$$A3 = \begin{pmatrix} a3_1 & a3_2 & \cdots & a3_{10} \end{pmatrix}$$

1.4.2.2 前向计算

用大写符号的矩阵形式的公式来描述，在每个矩阵符号的右上角是其形状。

隐层1

$$Z1 = X \cdot W1 + B1 \qquad (1)$$

$$A1 = Sigmoid(Z1) \qquad (2)$$

隐层2

$$Z2 = A1 \cdot W2 + B2 \qquad (3)$$

$$A2 = Tanh(Z2) \qquad (4)$$

输出层

$$Z3 = A2 \cdot W3 + B3 \qquad (5)$$

$$A3 = Softmax(Z3) \qquad (6)$$

约定是行为样本，列为一个样本的所有特征，这里是784个特征，因为图片高和宽均为28，总共784个点，把每一个点的值做为特征向量。

两个隐层，分别定义64个神经元和16个神经元。第一个隐层用Sigmoid激活函数，第二个隐层用Tanh激活函数。

输出层10个神经元，再加上一个Softmax计算，最后有$a1, a2, \ldots a10$共十个输出，分别代表0-9的10个数字。

1.4.2.3 反向传播

和以前的两层网络没有多大区别，只不过多了一层，而且用了tanh激活函数，目的是想把更多的梯度值回传，因为tanh函数比sigmoid函数稍微好一些，比如原点对称，零点梯度值大。

输出层

$$dZ3 = A3 - Y \quad (7)$$
$$dW3 = A2^{\top} \cdot dZ3 \quad (8)$$
$$dB3 = dZ3 \quad (9)$$

隐层2

$$dA2 = dZ3 \cdot W3^{\top} \quad (10)$$
$$dZ2 = dA2 \odot (1 - A2 \odot A2) \quad (11)$$
$$dW2 = A1^{\top} \cdot dZ2 \quad (12)$$
$$dB2 = dZ2 \quad (13)$$

隐层1

$$dA1 = dZ2 \cdot W2^{\top} \quad (14)$$
$$dZ1 = dA1 \odot A1 \odot (1 - A1) \quad (15)$$
$$dW1 = X^{\top} \cdot dZ1 \quad (16)$$
$$dB1 = dZ1 \quad (17)$$

1.4.3 梯度检查

1.4.3.1 为何要做梯度检查?

神经网络算法使用反向传播计算目标函数关于每个参数的梯度，可以看做解析梯度。由于计算过程中涉及到的参数很多，用代码实现的反向传播计算的梯度很容易出现误差，导致最后迭代得到效果很差的参数值。

为了确认代码中反向传播计算的梯度是否正确，可以采用梯度检验（gradient check）的方法。通过计算数值梯度，得到梯度的近似值，然后和反向传播得到的梯度进行比较，若两者相差很小的话则证明反向传播的代码是正确无误的。

1.4.3.2 算法

1. 初始化神经网络的所有矩阵参数（可以使用随机初始化或其它非0的初始化方法）
2. 把所有层的W, B都转化成向量，按顺序存放在θ中
3. 随机设置X值，最好是归一化之后的值，在[0,1]之间
4. 做一次前向计算，再紧接着做一次反向计算，得到各参数的梯度$d\theta_{real}$
5. 把得到的梯度$d\theta_{real}$变化成向量形式，其尺寸应该和第2步中的θ相同，且一一对应（W对应dW, B对应dB）
6. 对2中的θ向量中的每一个值，做一次双边逼近，得到$d\theta_{approx}$
7. 比较$d\theta_{real}$和$d\theta_{approx}$的值，通过计算两个向量之间的欧式距离：

$$diff = \frac{\|d\theta_{real} - d\theta_{approx}\|_2}{\|d\theta_{approx}\|_2 + \|d\theta_{real}\|_2}$$

结果判断：

1. $diff > 1e^{-2}$

   梯度计算肯定出了问题。

2. $1e^{-2} > diff > 1e^{-4}$

   可能有问题了，需要检查。

3. $1e^{-4} > diff > 1e^{-7}$

   不光滑的激励函数来说时可以接受的，但是如果使用平滑的激励函数如 tanh nonlinearities and softmax，这个结果还是太高了。

4. $1e^{-7} > diff$

   成功

另外要注意的是，随着网络深度的增加会使得误差积累，如果用了10层的网络，得到的相对误差为`1e-2`那么这个结果也是可以接受的。

1.4.3.3 注意事项

1. 首先，不要使用梯度检验去训练，即不要使用梯度检验方法去计算梯度，因为这样做太慢了，在训练过程中，我们还是使用backprop去计算参数梯度，而使用梯度检验去调试，去检验backprop的过程是否准确。
2. 其次，如果我们在使用梯度检验过程中发现backprop过程出现了问题，就需要对所有的参数进行计算，以判断造成计算偏差的来源在哪里，它可能是在求解B出现问题，也可能是在求解某一层的W出现问题，梯度检验可以帮助我们确定发生问题的范围，以帮助我们调试。
3. 别忘了正则化。如果我们添加了二范数正则化，在使用backprop计算参数梯度时，不要忘记梯度的形式已经发生了变化，要记得加上正则化部分，同理，在进行梯度检验时，也要记得目标函数J的形式已经发生了变化。
4. 注意，如果我们使用了drop-out正则化，梯度检验就不可用了。为什么呢？因为我们知道drop-out是按照一定的保留概率随机保留一些节点，因为它的随机性，目标函数J的形式变得非常不明确，这时我们便无法再用梯度检验去检验backprop。如果非要使用drop-out且又想检验backprop，我们可以先将保留概率设为1，即保留全部节点，然后用梯度检验来检验backprop过程，如果没有问题，我们再改变保留概率的值来应用drop-out。
5. 最后，介绍一种特别少见的情况。在刚开始初始化W和b时，W和b的值都还很小，这时backprop过程没有问题，但随着迭代过程的进行，W和B的值变得越来越大时，backprop过程可能会出现问题，且可能梯度差距越来越大。要避免这种情况，我们需要多进行几次梯度检验，比如在刚开始初始化权重时进行一次检验，在迭代一段时间之后，再使用梯度检验去验证backprop过程。

1.4.4 学习率与批大小

在梯度下降公式中：

$$w_{t+1} = w_t - \frac{\eta}{m}\sum_{i}^{m}\nabla J(w,b) \qquad (1)$$

其中，η是学习率，m是批大小。所以，学习率与批大小是对梯度下降影响最大的两个因子。

1.4.4.1 初始学习率的选择

Leslie N. Smith 在2015年的一篇论文Cyclical Learning Rates for Training Neural Networks中的描述了一个非常棒的方法来找初始学习率。

这个方法在论文中是用来估计网络允许的最小学习率和最大学习率，方法非常简单：

1. 首先设置一个非常小的初始学习率，比如`1e-5`；
2. 然后在每个`batch`之后都更新网络，计算损失函数值，同时增加学习率；
3. 最后可以描绘出学习率的变化曲线和loss的变化曲线，从中就能够发现最好的学习率。

1.4.4.2 学习率的后期修正

fixed

使用固定的学习率，比如全程都用0.1。要注意的是，这个值不能大，否则在后期接近极值点时不易收敛。

step

每迭代一个预订的次数后（比如500步），就调低一次学习率。离散型，简单实用。

multistep

预设几个迭代次数，到达后调低学习率。与step不同的是，这里的次数可以是不均匀的，比如3000、5500、8000。离散型，简单实用。

exp

连续的指数变化的学习率，公式为：

$$lr_{new} = lr_{base} * \gamma^{iteration} \tag{5}$$

由于一般的iteration都很大（训练需要很多次迭代），所以学习率衰减得很快。γ可以取值0.9、0.99等接近于1的数值，数值越大，学习率的衰减越慢。

inv

倒数型变化，公式为：

$$lr_{new} = lr_{base} * \frac{1}{(1+\gamma*iteration)^p} \tag{6}$$

γ控制下降速率，取值越大下降速率越快；p控制最小极限值，取值越大时最小值越小，可以用0.5来做缺省值。

poly

多项式衰减，公式为：

$$lr_{new} = lr_{base} * (1 - \frac{iteration}{iteration_{max}})^p \tag{7}$$

$p = 1$时，为线性下降；$p > 1$时，下降趋势向上突起；$p < 1$时，下降趋势向下凹陷。p可以设置为0.9。

1.1.4.3 学习率与批大小的关系

对此实际上是有两个建议：

1. 如果增加了学习率，那么batch size最好也跟着增加，这样收敛更稳定。
2. 尽量使用大的学习率，因为很多研究都表明更大的学习率有利于提高泛化能力。如果真的要衰减，可以尝试其他办法，比如增加batch size，学习率对模型的收敛影响真的很大，慎重调整。

batch size和学习率的关系可以大致总结如下：

1. 增加batch size，需要增加学习率来适应，可以用线性缩放的规则，成比例放大
2. 到一定程度，学习率的增加会缩小，变成batch size的$\sqrt{m}$倍
3. 到了比较极端的程度，无论batch size再怎么增加，也不能增加学习率了