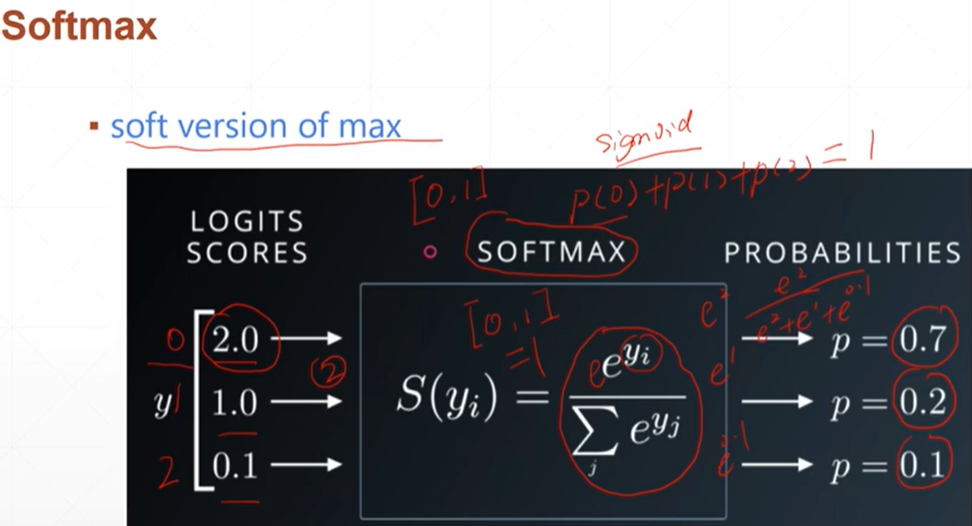
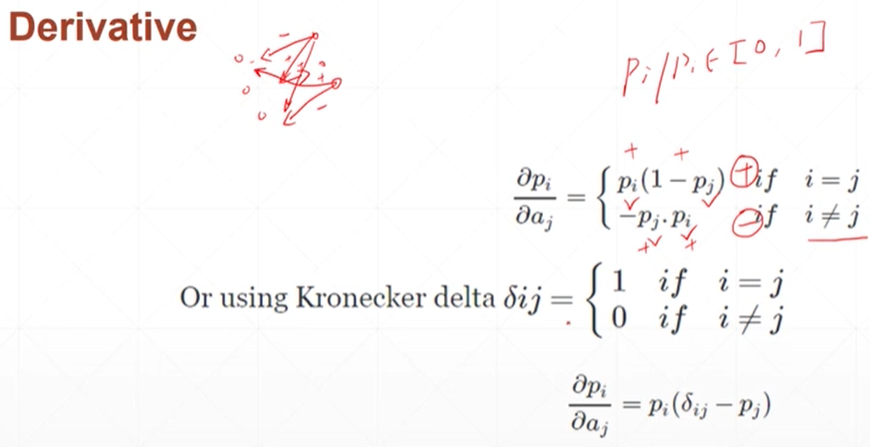
**pytorch—预测值转换为概率，单层感知机，多层感知机，分类问题**

**softmax函数，可以将算出来的预测值转换成0-1之间的概率形式**

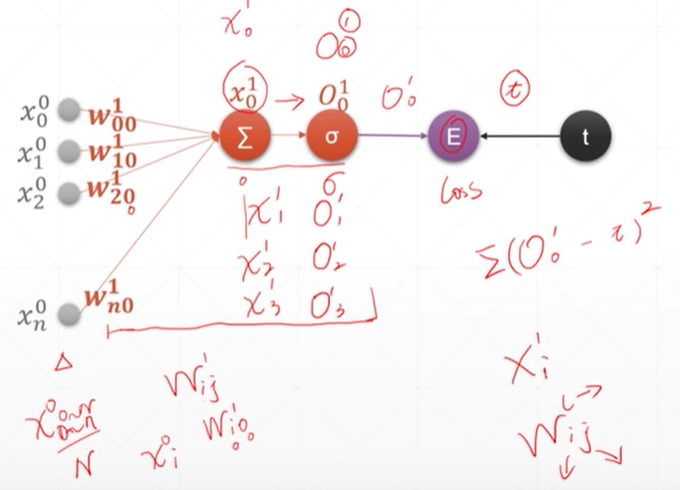


导数的形式



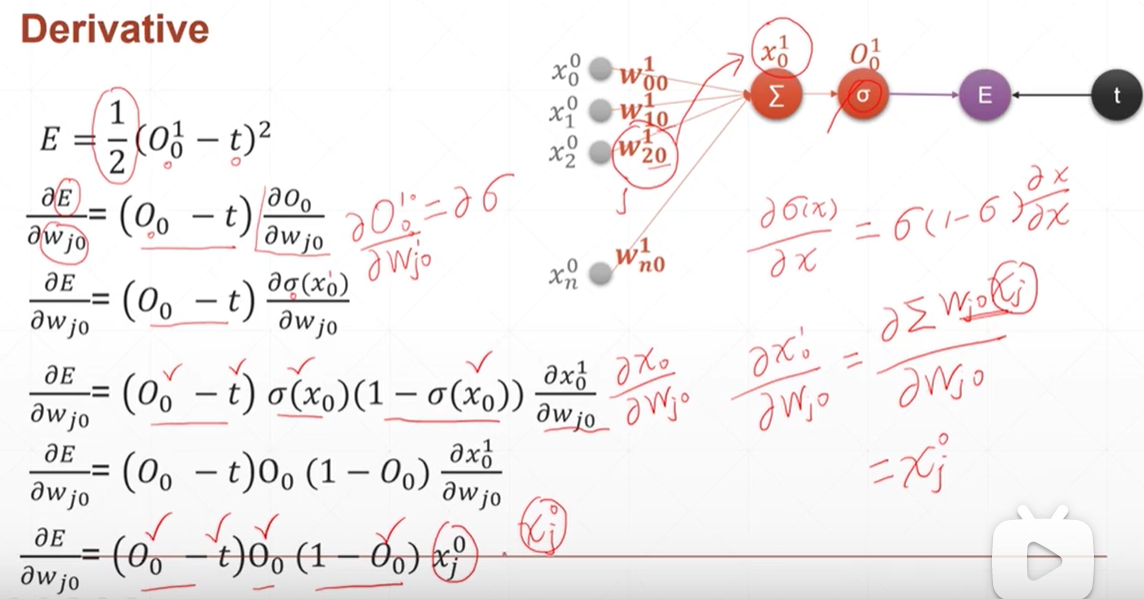
import torchimport torch.nn.functional as Fx=torch.tensor([3.3,2.2,1.0])x.requires\_grad\_()y=F.softmax(x,dim=0)print('将x转换成概率型的y',y)print(y[0],x[0])print('对y1进行求导,由于y是由所有xi来生成的，所以传输入的时候要把所有的x传进去')#由于y=0.6978,0.2323,0.07. 所以有导数公式dy1/dx1=0.6978\*(1-0.6978)=0.2109 dy1/dx2=-0.6978\*0.2323=-0.162print('y0对上x0-x3三个方向上的导数',torch.autograd.grad(outputs=y[0], inputs=x))y=F.softmax(x,dim=0)print('y1对上x0-x3三个方向上的导数',torch.autograd.grad(outputs=y[1], inputs=x))y=F.softmax(x,dim=0)print('y2对上x0-x3三个方向上的导数',torch.autograd.grad(outputs=y[2], inputs=x))

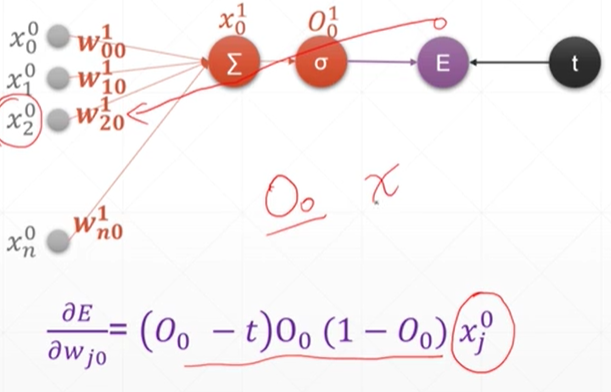
**单层感知机**



x的上标代表层数，下面的下标代表的是节点的编号。w的上标是下一层，下标的第一位是上一层的节点的编号，第二位是上一层

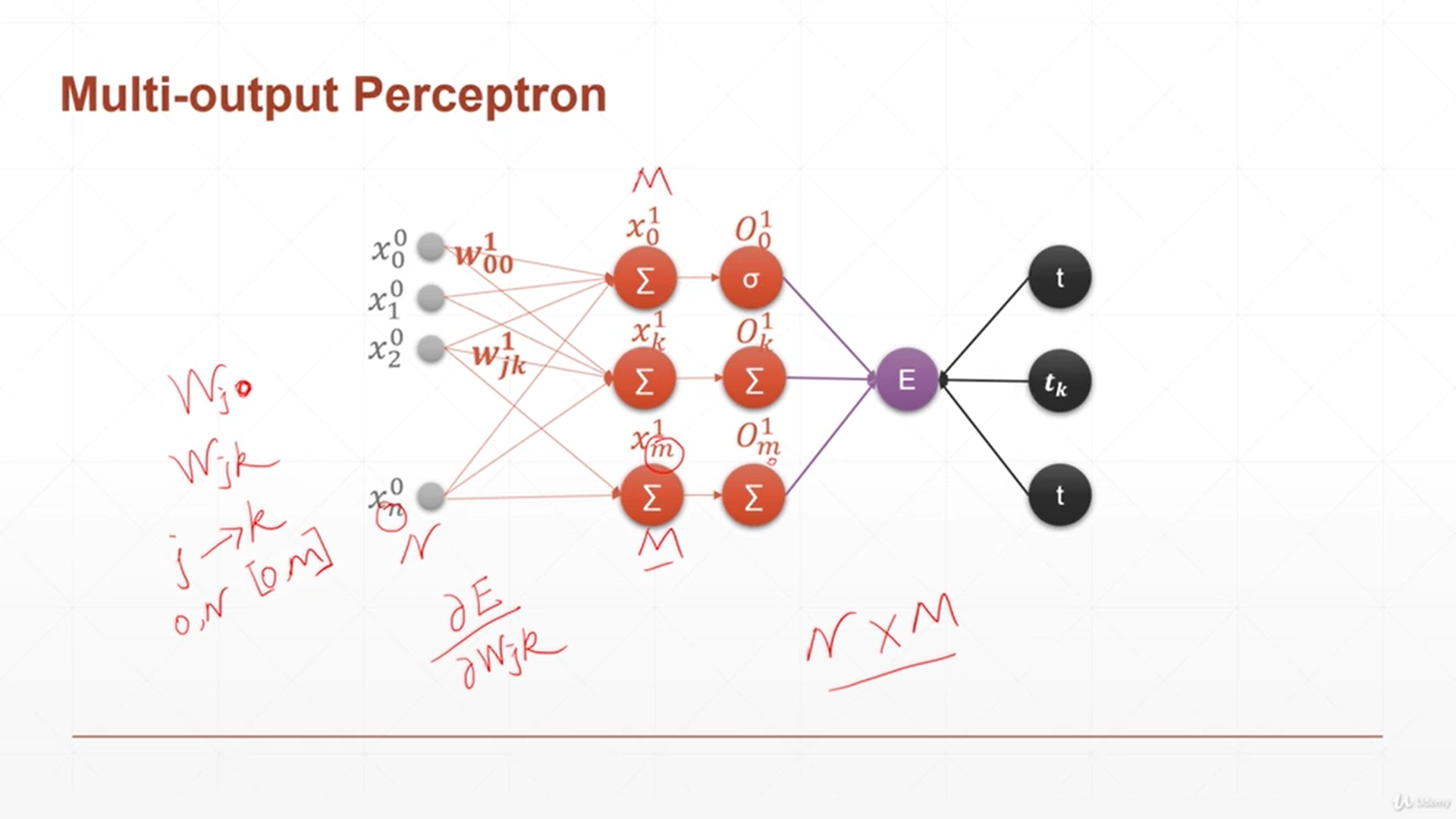
第0层的n个节点通过权值相乘再累加得到下一层的x，然后x通过激活函数再计算损失





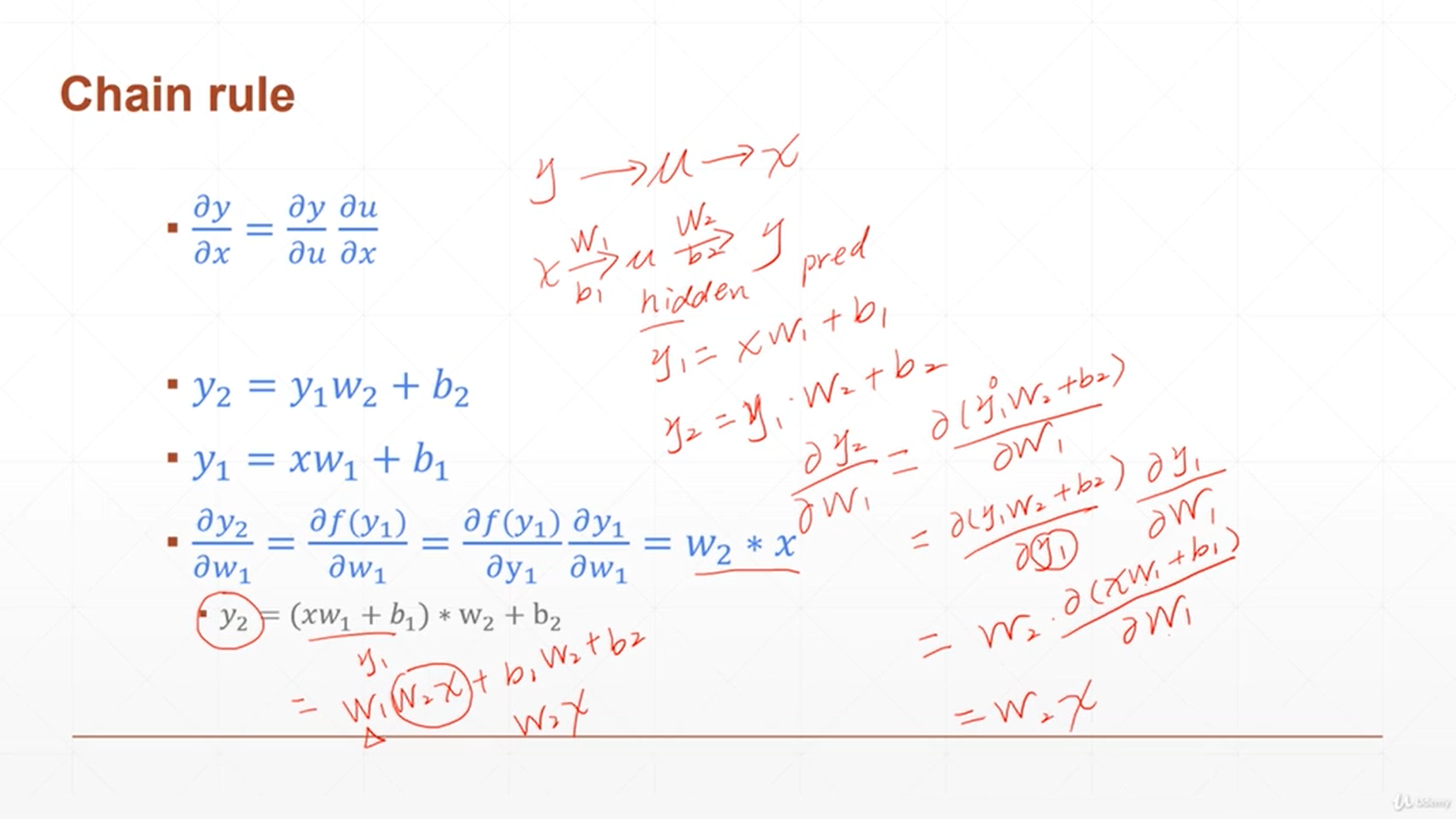
import torchimport torch.nn.functional as Fx=torch.rand(1,10)w=torch.rand(1,10)print('x向量',x,'w向量',w)w.requires\_grad\_()y=torch.sigmoid(x@w.t()) #是向量相乘然后求sigmoid函数变到下一层的神经元xcountshape=list(y.shape) #看它的shapeprint(y.shape,countshape)loss=F.mse\_loss(torch.ones(countshape),y) #用torch.ones(shape值)和下层神经元x(y)求lossprint(loss)# print(torch.ones(countshape))loss.backward()print(w.grad) #求得y对所有的wi梯度

**多层感知机**



#感知机的第二层有两个xx=torch.rand(1,10)w=torch.rand(2,10) #w有两维，也就是两组，意味着两组分别与xi构成两个下一层的print('x向量',x,'w向量',w) #这里是二维的，就说明w.requires\_grad\_()y=torch.sigmoid(x@w.t()) #是向量相乘然后求sigmoid函数变到下一层的神经元xcountshape=list(y.shape) #看它的shapeprint(y.shape,countshape)loss=F.mse\_loss(torch.ones(countshape),y) #用torch.ones(shape值)和下层神经元x(y)求lossprint(loss)# print(torch.ones(countshape))loss.backward()print(w.grad) #求得yi(i属于0--1)对所有的wj(j属于0--9)梯度

**链式法则**



x=torch.tensor(1.)w1=torch.tensor(2.)w1.requires\_grad\_()b1=torch.tensor(1.)w2=torch.tensor(2.)w2.requires\_grad\_()b2=torch.tensor(1.)y1=x\*w1+b1y2=y1\*w2+b2print('输入的x(%s)，经过w1(%s)和b1(%s),生成y1(%s).再通过w2(%s)和b2(%s)生成y2(%s)' %(x,w1,b1,y1,w2,b2,y2))#现实的情况一般是已知输入然后去求权重和偏置，因此最终要求的总梯度是dy2\_dw1dy2\_dy1=torch.autograd.grad(y2,[y1],retain\_graph=True)[0]dy1\_dw1=torch.autograd.grad(y1,[w1],retain\_graph=True)[0]print(dy2\_dy1\*dy1\_dw1)dy2\_dw1=torch.autograd.grad(y2,[w1],retain\_graph=True)[0]print(dy2\_dw1)

**函数极小值的优化**

1.用plt画出图像

from matplotlib import pyplot as pltfrom mpl\_toolkits.mplot3d import Axes3Dimport numpy as npdef fomul(x): return (x[0]\*\*2+x[1]-11)\*\*2+(x[0]+x[1]\*\*2-7)\*\*2x=np.arange(-6,6,0.1)y=np.arange(-6,6,0.1)X,Y=np.meshgrid(x,y) #这一步是制作出x和y的网格化矩阵print([X,Y]) #这一步是两个网格化矩阵放进列表Z=fomul([X,Y]) #将x的网格化矩阵和y的网格化矩阵放进公式计算出z的网格化矩阵fig = plt.figure()ax = plt.gca(projection='3d')ax.plot\_surface(X,Y,Z)ax.view\_init(60,-30)plt.show()

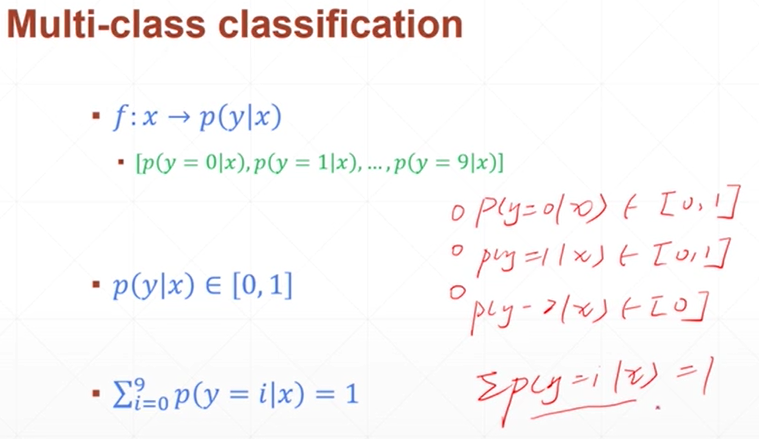
**逻辑回归**

使用sigmore函数转变成概率

liner regression 和 logistic regression的区别:

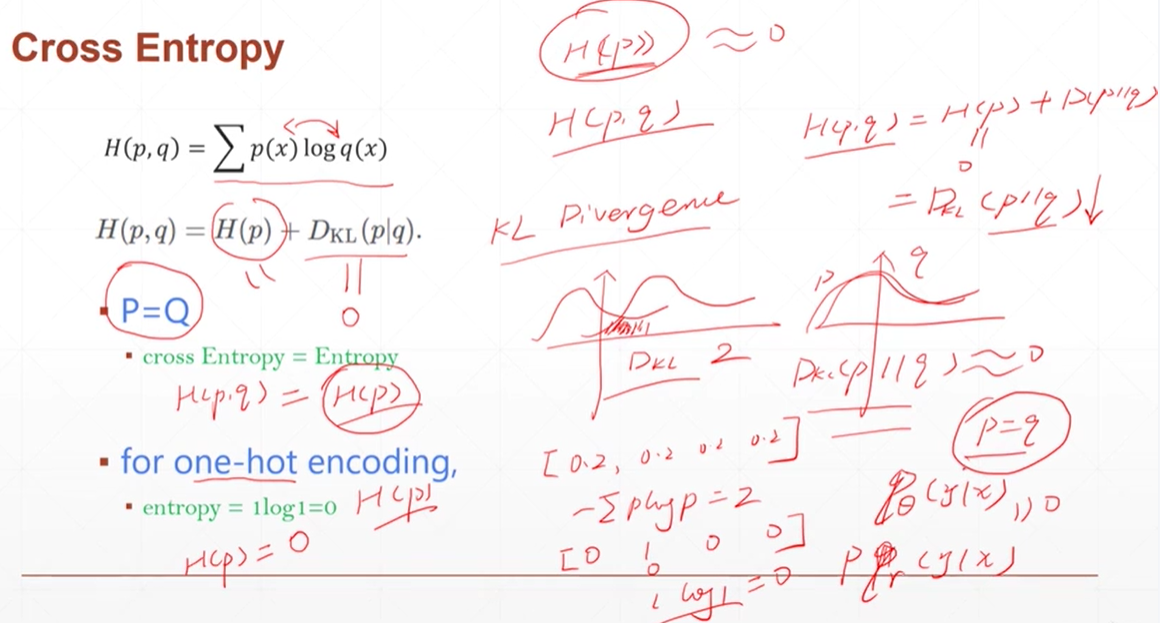
1. 输出的区别. linear regression的输出是连续的,在有限空间可取任意值; logistic regression的输出期望是离散的,只有有限个数值.
2. 预期目标(label)的区别. linear regression的预期是连续变量,如auto-encoder模型预测一张图像; logistir regression的预期是离散的类别.
3. 最小化误差的方法区别. 采用均方误差的linear regression对于大的误差施加二次倍数的惩罚, 而logistic regression把较大的误差惩罚到一个渐进的常数.
4. 先验的区别.liner regression期望拟合训练数据,通过feature的线性加权来预测结果; logistic regression是在训练一个最大似然分类器

**多分类问题**



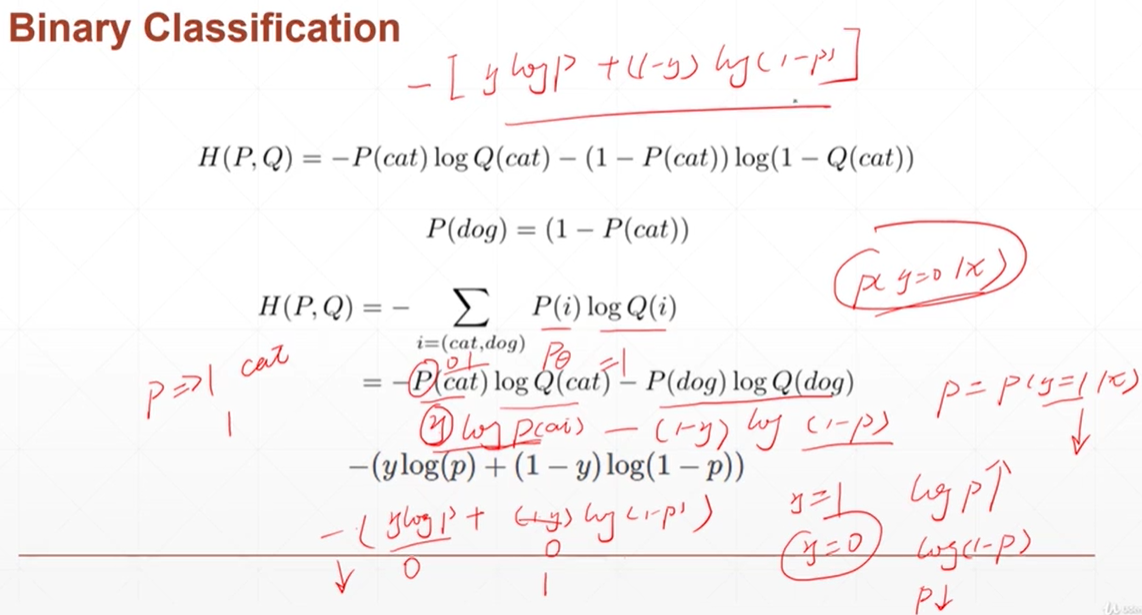
要将输出值转化到多分类的概率可以使用之前说的softmax函数来转换

**分类问题中常用的loss损失：cross entropy（交叉熵）**

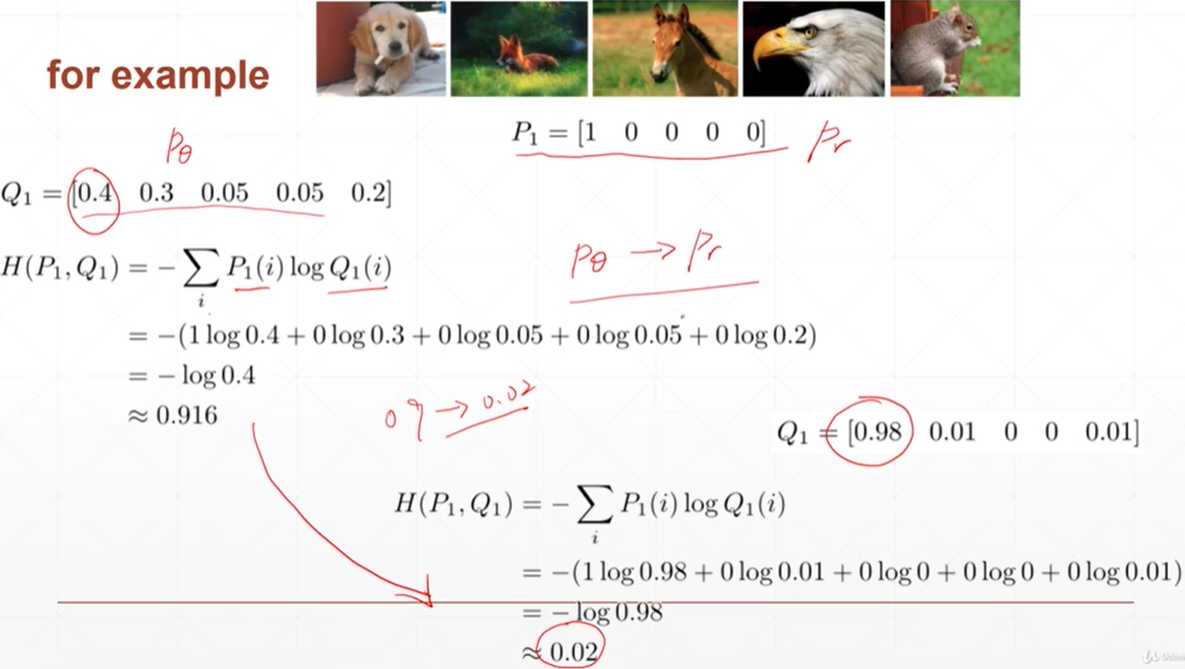


Dkl指的是散度，散开的程度，当p和q相等的时候重合的时候，散度是接近于0的因此H(p,q)就只等于H(p)

对于01编码的情况时,[0 1 0 0], 熵=-1log1=0，此时交叉熵等于散度



二分类问题的公式推导。p的概率是预测标签为1的概率。当标签y=0的时候，相应的log(1-p)大也就是标签1的概率要大。当标签y=1的时候，相应的log(p)大也就是标签0的概率要大。



分类的目的就是原本的q1的几个概率没有逼近1的，此时交叉熵也比较大。经过不断的迭代q1里面出现有一个概率是非常接近1的，而且它的交叉熵此时会变得非常小。自然预测值就是那个概率最大的。