计算机视觉复习课

神经网络的前后向传播

均方误差（损）失函数：

$$J = -\frac{1}{m}\sum_{j=1}^{m}(y^{(j)} - a^{(j)})^2$$

交叉熵代价（损失）函数：

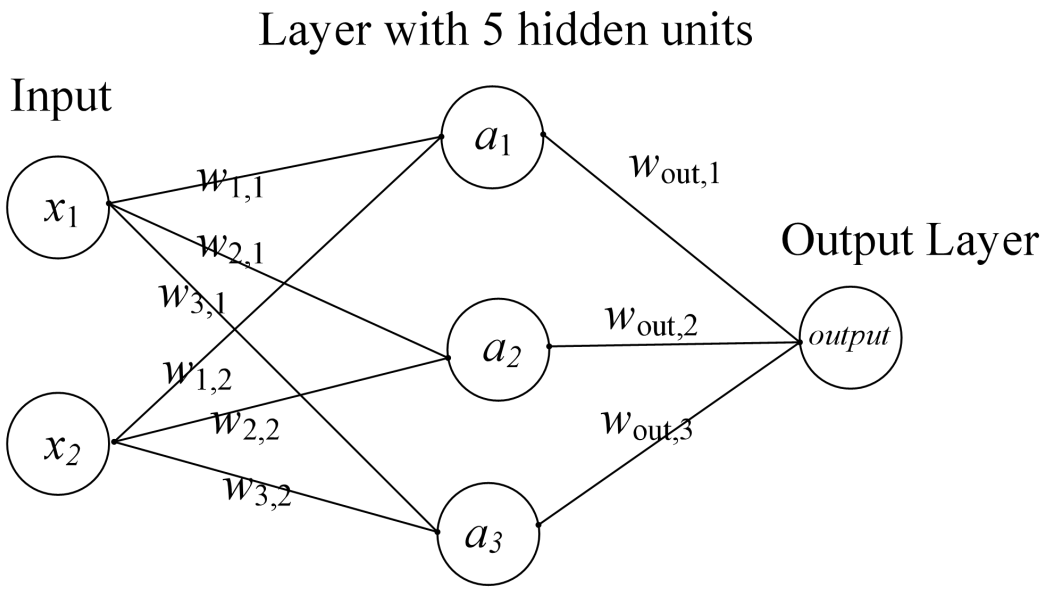
$$J = -\frac{1}{m}\sum_{j=1}^{m}(y^{(j)} \cdot log\ a^{(j)} + (1 - y^{(j)}) \cdot log(1 - a^{(j)}))$$

神经网络的前后向传播

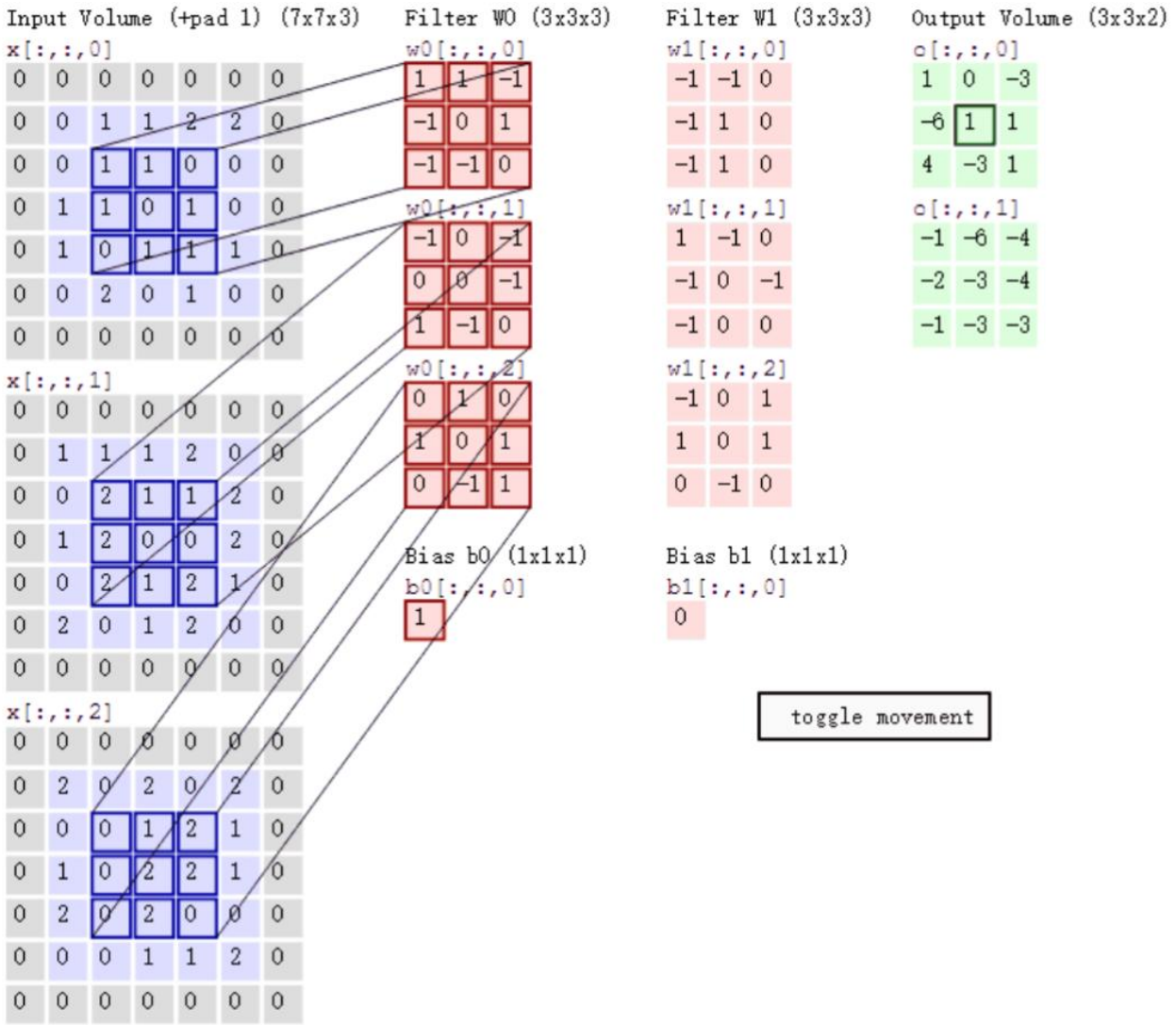
激活函数：

sigmoid函数： $f(x)=\frac{1}{1+e^{-x}}$

ReLu函数： $f(x)=max(0,x)$



卷积神经网络


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 1 2 2 0
0 0 1 1 0 0 0
0 1 1 0 1 0 0
0 1 0 1 1 1 0
0 0 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 0 2 1 1 2 0
0 1 2 0 0 2 0
0 0 2 1 2 1 0
0 2 0 1 2 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 0 2 0 2 0
0 0 0 1 2 1 0
0 1 0 2 2 1 0
0 2 0 2 0 0 0
0 0 0 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 -1
-1 0 1
-1 -1 0
w0[:,:,1]
-1 0 -1
0 0 -1
1 -1 0
w0[:,:,2]
0 1 0
1 0 1
0 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 0
-1 1 0
w1[:,:,1]
1 -1 0
-1 0 -1
-1 0 0
w1[:,:,2]
-1 0 1
1 0 1
0 -1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
1 0 -3
-6 1 1
4 -3 1
o[:,:,1]
-1 -6 -4
-2 -3 -4
-1 -3 -3
toggle movement

pytorch搭建卷积神经网络

参考实验3

```
class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1=torch.nn.Sequential(
        torch.nn.Conv2d(1,64,kernel_size=3,stride=1,padding=1),
        torch.nn.ReLU(),
        torch.nn.Conv2d(64,128,kernel_size=3,stride=1,padding=1),
        torch.nn.ReLU(),
        torch.nn.MaxPool2d(stride=2,kernel_size=2))
        self.dense=torch.nn.Sequential(
        torch.nn.Linear(14*14*128,1024),
        torch.nn.ReLU(),
        torch.nn.Dropout(p=0.5),
        torch.nn.Linear(1024, 10))

    def forward(self, x):
        x = self.conv1(x)  # 卷积处理
        x = x.view(-1, 14*14*128)  # 对参数实行扁平化处理
        x = self.dense(x)
        return x
```

迁移学习

为什么需要迁移学习

数据的角度

- 收集数据很困难
- 为数据打标签很耗时
- 训练一对一的模型很繁琐

模型的角度

- 个性化模型很复杂
- 云+端的模型需要作具体化适配

应用的角度

冷启动问题：没有足够用户数据，推荐系统无法工作

因此，迁移学习是必要的

迁移学习应用场景

应用前景广阔

模式识别、计算机视觉、语音识别、自然语言处理、数据挖掘...

语料匮乏条件下不同语言的相互翻译学习

不同视角、不同背景、不同光照的图像识别

不同用户、不同设备、不同位置的行为识别

不同领域、不同背景下的文本翻译、舆情分析

不同用户、不同接口、不同情境的人机交互

不同场景、不同设备、不同时间的室内定位

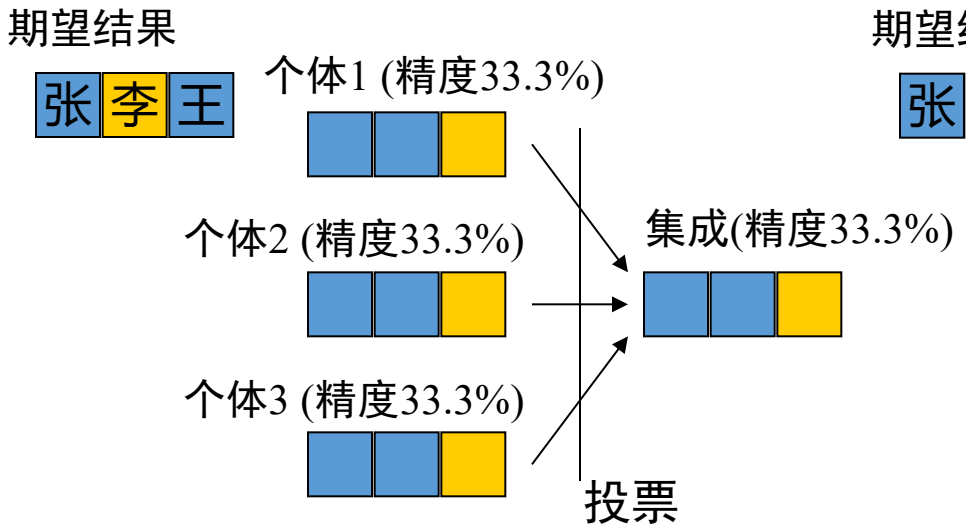
模型迁移学习

参考P186页

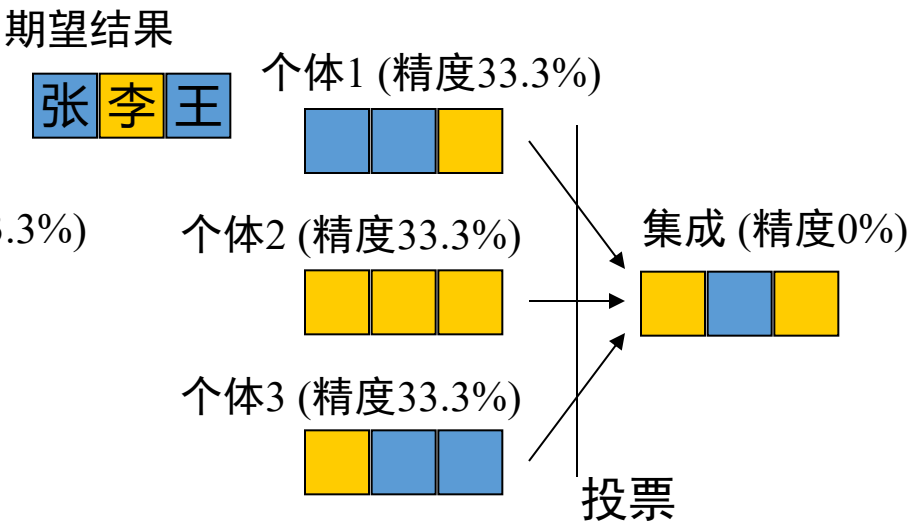
以下是在cifa100（100个类别）数据集上训练的VGG网络模型，将上述网络模型迁移到minist数据集（10个类别）的字符识别任务中，如何修改？
分别分析以下操作：
'conv1'，'relu1'，'relu2'，'pool1'，'conv3'，'relu3'，'relu3'，'conv4'，'relu4'，'pool2'，'fc1'，'relu5'，输入输出的数据维度大小。

```
class Models(torch.nn.Module):
    def __init__(self):
        super(Models, self).__init__()
        self.Conv = torch.nn.Sequential(OrderedDict([
            ('conv1',torch.nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)),
            ('relu1',torch.nn.ReLU()),
            ('conv2',torch.nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1)),
            ('relu2',torch.nn.ReLU()),
            ('pool1',torch.nn.MaxPool2d(kernel_size=2, stride=2)),
            ('conv3',torch.nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)),
            ('relu3',torch.nn.ReLU()),
            ('conv4',torch.nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1)),
            ('relu4',torch.nn.ReLU()),
            ('pool2',torch.nn.MaxPool2d(kernel_size=2, stride=2)),
        self.Classes = torch.nn.Sequential(
            ('fc1',torch.nn.Linear(8*8*128, 1024)),
            ('relu5',torch.nn.ReLU()),
            ('fc2',torch.nn.Linear(1024, 100))
        )
```

集成学习系统的构建



个体必须有差异



个体精度不能太低

$$E = \overline{E} - \overline{A}$$ [A. Krogh & J. Vedelsby, NIPS94]

个体学习器越精确、差异越大，集成越好

多模型融合

P234页-238页

多个模型的性能如下图

模型一的预测结果：									
80%	70%	70%	70%	70%	70%	70%	70%	20%	40%
模型二的预测结果：									
30%	46%	70%	70%	70%	70%	70%	70%	70%	70%
模型三的预测结果：									
70%	40%	70%	40%	70%	70%	70%	70%	30%	70%

多模型融合

平均和加权平均融合：

模型一的预测结果：									
80%	70%	70%	70%	70%	70%	70%	70%	20%	40%
模型二的预测结果：									
30%	46%	70%	70%	70%	70%	70%	70%	70%	70%
模型三的预测结果：									
70%	40%	70%	40%	70%	70%	70%	70%	30%	70%

平均	**0.6**								
加权平均 (5：3：2)	0.56								

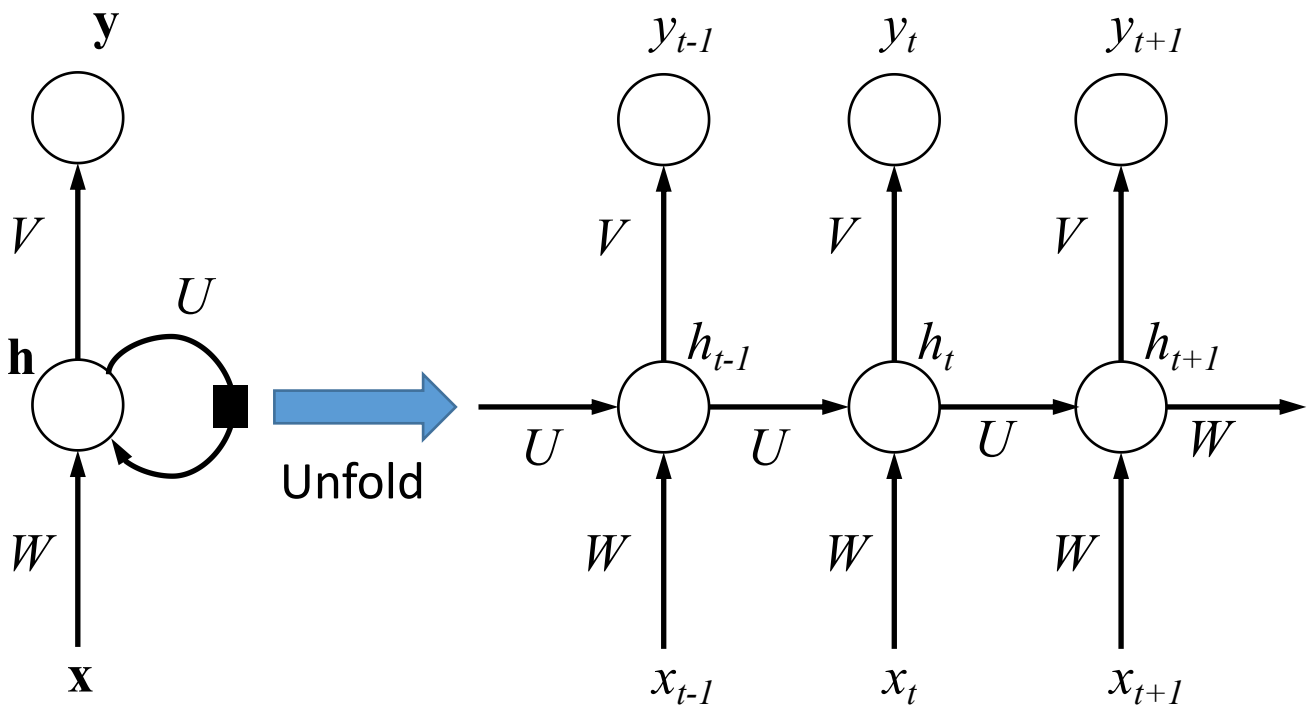
多模型融合

多模型投票融合：
1.将概率转化为判别预测结果；
2.进行少数服从多数统计。

模型一的预测结果：									
True	True	True	True	True	True	True	True	False	False
模型二的预测结果：									
False	False	True	True	True	True	True	True	True	True
模型三的预测结果：									
True	False	True	False	False	True	True	True	False	True

循环神经网络

循环神经网络（Recurrent Neural Network，RNN）是一种对序列数据建模的神经网络，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不在无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。RNN模型的连接如图所示。



循环神经网络

损失函数分析：P250页、P251页

```
class RNN(torch.nn.Module):
    def __init__(self):
        super(RNN, self).__init__()
        self.rnn = torch.nn.RNN(
            input_size = 28,
            hidden_size = 128,
            num_layers = 1,
            batch_first = True
        )
        self.output = torch.nn.Linear(128,10)

    def forward(self, input):
        output,_ = self.rnn(input, None)
        output = self.output(output[:,-1,:])
        return output
```

在以上代码中，我们定义的input_size 为28，这是因为输入的手写数据的宽高为28×28，所以可以将每一张图片看作序列长度为28 且每个序列中包含28 个数据的组合。模型最后输出的结果是用作分类的，所以仍然需要输出10 个数据，在代码中的体现就是self.output= torch.nn.Linear(128,10)。再来看前向传播函数forward 中的两行代码，首先是output,_ =self.rnn(input, None)，其中包含两个输入参数，分别是input 输出数据和H0 的参数。在循环神经网络模型中，对H0 的初始化我们一般采用0 初始化，所以这里传入的参数就是None。再看代码output = self.output(output[:,−1,:])，因为我们的模型需要处理的是分类问题，所以需要提取最后一个序列的输出结果作为当前循环神经网络模型的输出。

自编码器

数据制作过程：P259页，260页

本节的自动编码器模型解决的是一个去除图片马赛克的问题。要训练出这个模型，我们首先需要生成一部分有马赛克的图片，实现图片打码操作的代码如下：

```
noisy_images = images+ 0.5*np.random.randn(*images.shape)
noisy_images = np.clip(noisy_images, 0., 1.)
```

以上代码中的images 是我们现有的正常图片，我们知道图片是由像素点构成的，而像素点其实就是一个个的数字，我们使用的MNIST 数据集中的手写图片的像素点数字的范围是0 到1，所以处理马赛克的一种简单方式就是对原始图片中的像素点进行扰乱，我们在这里通过对输入的原始图片加上一个维度相同的随机数字来达到了处理马赛克的目的。假设我们原始的输入图片如图11-2 所示。则经过我们的打码处理后，图片如图11-3 所示。

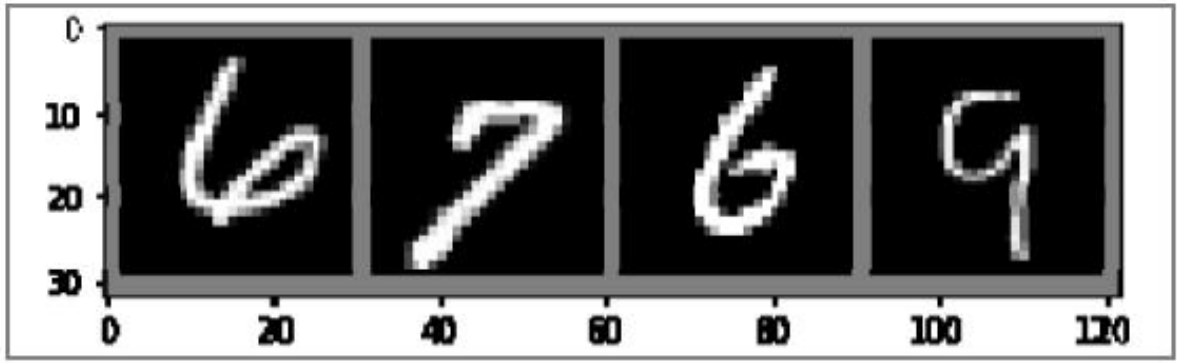


图 11-2

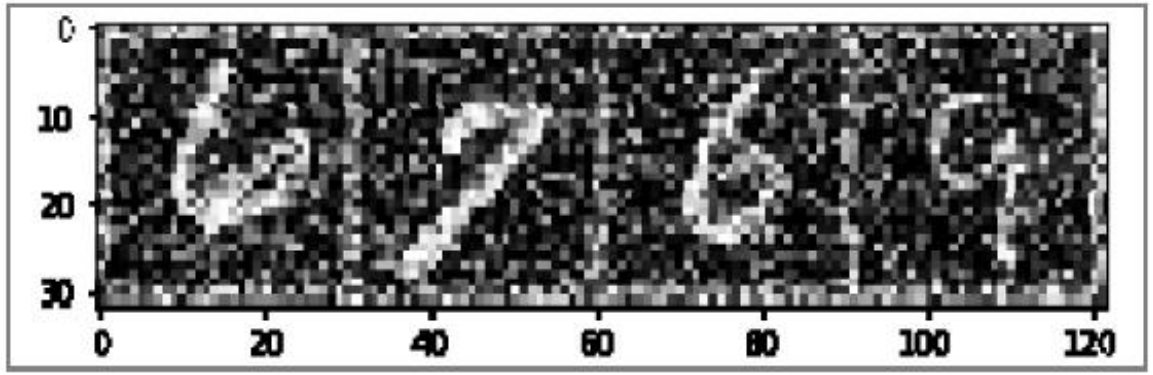


图 11-3

自编码器

损失函数分析：P262页

自编码器的损失函数使用的是torch.nn.MSELoss，即计算的是均方误差，我们在之前处理的都是图片分类相关的问题，所以在这里使用交叉熵来计算损失值。而在这个问题中我们需要衡量的是图片在去码后和原始图片之间的误差，所以选择均方误差这类损失函数作为度量。

均方误差代价（损）失函数：

$$J = -\frac{1}{m}\sum_{j=1}^{m}(y^{(j)} - a^{(j)})^2$$

交叉熵代价（损失）函数：

$$J = -\frac{1}{m}\sum_{j=1}^{m}(y^{(j)} \cdot log\ a^{(j)} + (1 - y^{(j)}) \cdot log(1 - a^{(j)}))$$