搭建一个神经网络

这一部分会不用PyTorch搭建一个神经网络并训练它。但是为了代码的简洁，我们会用到Numpy模块用于基本的矩阵计算。

In [1]:
```
import numpy as np
```

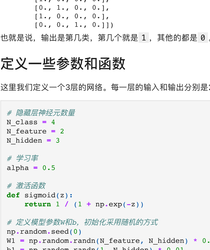
数据集

这里我们生成一个简单的数据集，输入2个维度，输出4类。每个象限是一个不同的类（包含噪音，比如有有少量的点，明明是第一类，它们的x_2却是负的，在现实中这往往由于测量误差造成）。

In [2]:
```
# 定义数据点的个数和数据的分散程度
N = 200
var = 0.5

# 第一个象限的数据
x1 = np.random.randn(N, 1) * var + 1
y1 = np.random.randn(N, 1) * var + 1
l1 = np.random.randn(N, 1) * 0 + 0
# 第二个象限的数据
x3 = np.random.randn(N, 1) * var - 1
y3 = np.random.randn(N, 1) * var + 1
l3 = np.random.randn(N, 1) * 0 + 1
# 第三个象限的数据
x4 = np.random.randn(N, 1) * var - 1
y4 = np.random.randn(N, 1) * var - 1
l4 = np.random.randn(N, 1) * 0 + 2
# 第四个象限的数据
x2 = np.random.randn(N, 1) * var + 1
y2 = np.random.randn(N, 1) * var - 1
l2 = np.random.randn(N, 1) * 0 + 3

# 把数据拼到一起
X1 = np.vstack((x1, x2, x3, x4))
X2 = np.vstack((y1, y2, y3, y4))
X = np.hstack((X1, X2))
# 把标签拼在一起
y = np.vstack((l1, l2, l3, l4))

# 把数据打乱
idx = np.random.permutation(X.shape[0])
X = X[idx,:]
y = y[idx,:]
```

展示几个数据，不同的颜色表示不同的类。

In [3]:
```
import matplotlib.pyplot as plt
plt.figure()
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



One-Hot Encoding

可以看出， `y` 是用整数表示的类。但是我们往往用one-hot encoding的方式编码这些信息：

In [4]:
```
def OneHot(y, K):
    E = y.flatten().shape[0]
    Y = np.zeros([E, K])
    for i in range(E):
        Y[i, int(y.flatten()[i])] = 1
    return Y
```

编码后的 `y` 就变成了

In [5]:
```
Y = OneHot(y, 4)
Y[:5,:]
```

Out[5]:
```
array([[0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 1., 0.]])
```

也就是说，输出是第几类，第几个就是 `1` ，其他的都是 `0` 。

定义一些参数和函数

这里我们定义一个3层的网络。每一层的输入和输出分别是2-3， 3-3， 3-4。一般简略的记为这是一个2-3-3-4的网络。

In [6]:
```
# 隐藏层神经元数量
N_class = 4
N_feature = 2
N_hidden = 3

# 学习率
alpha = 0.5

# 激活函数
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# 定义模型参数W和b，初始化采用随机的方式
np.random.seed(0)
W1 = np.random.randn(N_feature, N_hidden) * 0.01
b1 = np.random.randn(1, N_hidden) * 0.01
W2 = np.random.randn(N_hidden, N_hidden) * 0.01
b2 = np.random.randn(1, N_hidden) * 0.01
W3 = np.random.randn(N_hidden, N_class) * 0.01
b3 = np.random.randn(1, N_class) * 0.01
```

定义神经网络。这里我们定义一次训练的前向传播+后向传播+梯度下降

这里用到的公式和我们之前推导的一样。

In [7]:
```
def NN(X, Y, W1, W2, W3, b1, b2, b3, alpha):
    # 数据的数量
    E = X.shape[0]

    # 正向传播
    A0 = X

    Z1 = np.matmul(A0, W1) + b1
    A1 = sigmoid(Z1)

    Z2 = np.matmul(A1, W2) + b2
    A2 = sigmoid(Z2)

    Z3 = np.matmul(A2, W3) + b3
    A3 = sigmoid(Z3)

    # 损失函数
    L = np.sum(np.sum(-Y * np.log(A3) - (1 - Y) * np.log(1 - A3))) / E

    # 反向传播
    dZ3 = (A3 - Y) / E
    dW3 = np.matmul(A2.T, dZ3)
    db3 = np.sum(dZ3, axis=0, keepdims=True)

    dZ2 = np.matmul(dZ3, W3.T) * A2 * (1 - A2)
    dW2 = np.matmul(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)

    dZ1 = np.matmul(dZ2, W2.T) * A1 * (1 - A1)
    dW1 = np.matmul(A0.T, dZ1)
    db1 = np.sum(dZ1, axis=0, keepdims=True)

    # 梯度下降
    W1 = W1 - alpha * dW1
    W2 = W2 - alpha * dW2
    W3 = W3 - alpha * dW3

    b1 = b1 - alpha * db1
    b2 = b2 - alpha * db2
    b3 = b3 - alpha * db3

    # 返回新的参数，以及当前的损失函数
    return W1, W2, W3, b1, b2, b3, L
```

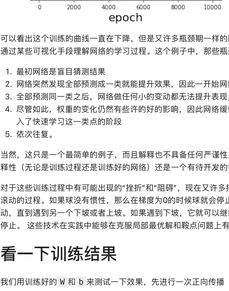
训练神经网络

这里进行10000次训练

In [8]:
```
Loss = []
for i in range(10000):
    W1, W2, W3, b1, b2, b3, L = NN(X, Y, W1, W2, W3, b1, b2, b3, alpha)
    Loss.append(L)
```

绘制loss在训练过程中的变化

In [9]:
```
plt.figure()
epochs = np.arange(len(Loss))
plt.plot(epochs, np.array(Loss))
plt.xlabel('epoch', fontsize=20)
plt.ylabel('loss', fontsize=20)
plt.show()
```



可以看出这个训练的曲线一直在下降，但是又许多瓶颈期一样的阶段。这其实就是神经网络难以解释的例子。但是对于简单的例子，人们或许可以通过某些可视化手段理解网络的学习过程。这个例子中，那些瓶颈期和下降期的原因是这样的：

1. 最初网络是盲目猜测结果
2. 网络突然发现全部预测成一类就能提升效果，因此一开始网络就往预测同一类开始变化，这就是开始的骤降
3. 全部预测同一类之后，网络做任何小的变动都无法提升表现，因此僵持住了（第一个平的阶段）
4. 尽管如此，权重的变化仍然有些许的好的影响，因此网络缓慢变化，直到发现：当把某一个点预测另外一类时，loss有明显下降，因此网络又进入了快速学习这一类点的阶段
5. 依次往复。

当然，这只是一个最简单的例子，而且解释也不具备任何严谨性，只是以人类的思维方式通过观察网络学习的过程去理解。真正的神经网络的可解释性（无论是训练过程还是训练好的网络）还是一个有待开发的领域。

对于这些训练过程中有可能出现的“挫折”和“阻碍”，现在又许多技术来应对。例如把“惯性”加到优化的过程中：让我们想象梯度下降是一个球往山下滚动的过程。如果球没有惯性，那么在梯度为0的时候球就会停止运动。某些技术会给这个球加上惯性，那么球在梯度为0的地方就会继续往前滚动，直到遇到另一个下坡或者上坡，如果遇到下坡，它就可以继续往下滚动，如果遇到上坡，它就会滚回来，然后在平坦的地方震荡，直到衰减到停止。 这些技术在实践中能够在充服局部最优解和鞍点问题上有很好的表现。后面还会提到，这里就不展开讲了。

看一下训练结果

我们用训练好的 `W` 和 `b` 来测试一下效果，先进行一次正向传播

In [10]:
```
Z1 = np.matmul(X, W1) + b1
A1 = sigmoid(Z1)
Z2 = np.matmul(A1, W2) + b2
A2 = sigmoid(Z2)
Z3 = np.matmul(A2, W3) + b3
A3 = sigmoid(Z3)
```

然后挑出A3中最大的值的位置作为预测的分类（比如第2个最大，就当作是第2类）

In [11]:
```
y_hat = np.argmax(A3, axis=1)
```

比较 `y_hat` 和 `y` ，注意这个 `y` 是没有one-hot编码的。如果一样就是 `True` ，不一样就是 `False` 。然后求和。求和的时候会自动把 `True` 当成1， `False` 当成0。这样，和就是正确的分类的个数。

In [12]:
```
corrects = np.sum(y_hat == np.argmax(A3, axis=1))
corrects
```

Out[12]:
```
800
```

然后再计算正确率，用正确的个数除以总的个数：

In [13]:
```
accuracy = corrects / X.shape[0]
accuracy
```

Out[13]:
```
1.0
```