机器学习-第九讲课后作业

姓名：周宝航 学号：2120190442 专业：计算机科学与技术

一、问题描述

已知：变量 x 在区间(0,1)上均匀采样获得数据集合$\{x_n\}$，其中 n=200；目标值t_n通过公式$x_n + 0.3\sin(2\pi x_n) + \varepsilon$计算得到，其中噪声$\varepsilon$均匀分布于区间(-0.1,0.1)。问题：数据集不变，使用混合密度网络求解映射$t_n \to x_n$。

1. 生成整个数据集，并绘制数据集的散布图。
2. 给出网络结构描述。
3. 给出网络训练算法描述。
4. 给出解及描述（对应课件中解的示意图）。
5. 实现源代码。

二、基本思路

1. 已给定变量 x 是在区间(0,1)上的均匀分布，而目标值t_n通过公式$x_n + 0.3\sin(2\pi x_n) + \varepsilon$计算得到，其中噪声$\varepsilon$是在区间(-0.1,0.1)上的均匀分布。我们采样得到变量 x 后通过公式计算得到对应的目标值 t。具体实现代码如下：

```python
# 生成数据
data_size = 200
x = np.random.uniform(size=data_size)
epsilon = np.random.uniform(low=-0.1, high=0.1, size=data_size)
t = x + 0.3 * np.sin(2*np.pi*x) + epsilon
```

```
# 绘制数据散布图
plt.scatter(t, x)
plt.show()
```

具体生成的样本数据如图 1 所示，由于题目需要求解$t_n \rightarrow x_n$的映射，我们设置图的横坐标为 t，纵坐标为 x。

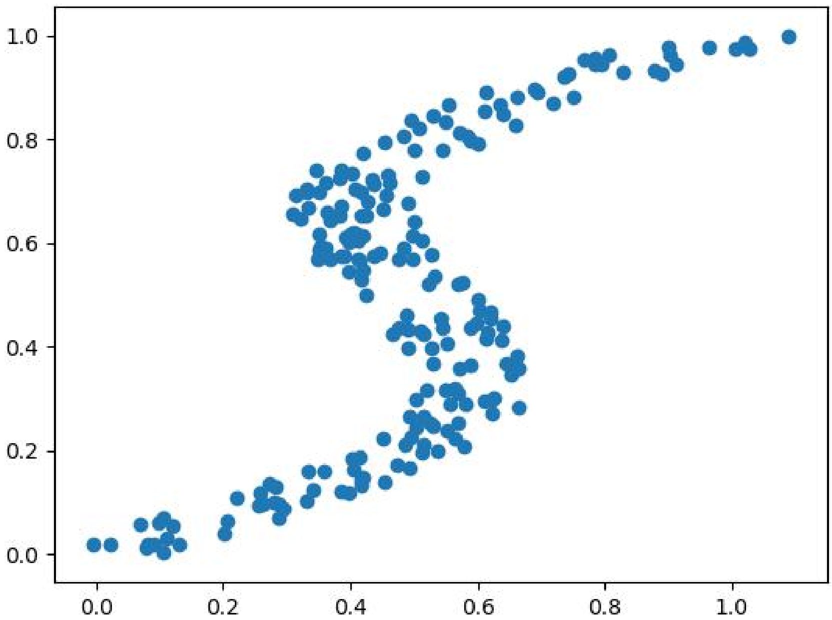

图 1 样本数据散布图

2. 混合密度网络是一种可以求解一对多映射问题的模型。该网络与普通网络不同，其输入矢量 x 进入网络后输出一个混合高斯分布。混合密度网络是一个两层神经网络，且隐含层单元使用 tanh 激活函数。如果混合模型有 L 个组成成分，目标变量有 K 个组成，那么网络将(K+2)L 个输出，包括：各分布的均值、方差以及混合系数。在训练过程中，我们采用前向传播算法获得预测分布，并根据目标值与预测值计算误差。然后采用反向传播算法计算每层参数的梯度，利用随机梯度下降方法更新权重与偏置。

三、解题步骤

1. 算法描述

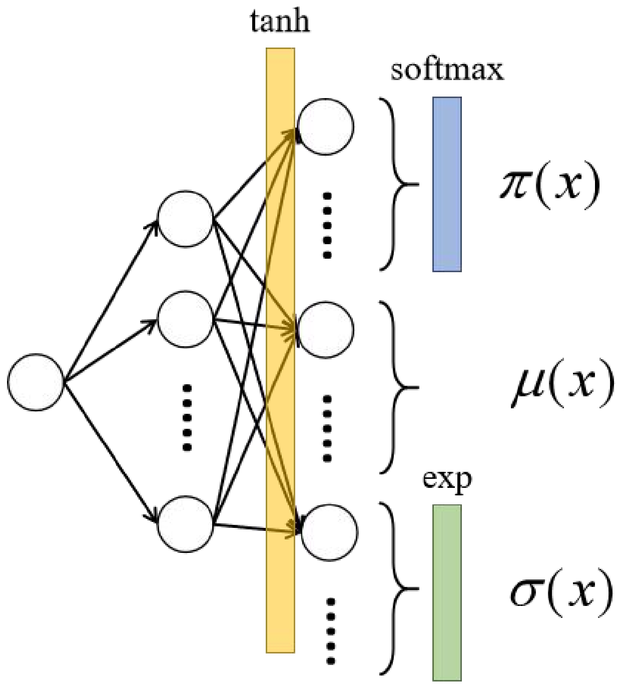

图 2 混合密度网络结构

网络由一个隐含层和一个输出层组成，其中隐含层使用 tanh 激活函数而输出层需要确定混合分布的不同参数。所以输出层采用图 2 所示对应的激活函数获取分布的不同参数。在本次实验中，我们假设混合成分为：3（由三种不同的高斯分布组成），目标变量有 1 个组成，那么我们的网络最终的输出层可以划分为：前 3 个神经元输出为混合系数 $\pi(x)$，中间 3 个神经元输出为均值 $\mu(x)$，最后 3 个神经元输出为方差 $\sigma(x)$。这些参数分别对应各个组成成分的分布参数。

混合系数必须满足：$\sum_{k=1}^{K}\pi_k(x)=1,\quad 0\le\pi_k\le 1$，所以我们采用 softmax 激活函数处理前三个神经元的净输出值：$\pi_k(x)=\frac{\exp(a_k^\pi)}{\sum_{l=1}^{K}\exp(a_l^\pi)}$。而方差必

须满足约束$\sigma_k^2(x)\ge 0$，表示为：$\sigma_k(x)=\exp(a_k^\sigma)$。我们最终定义误差函数为：$E(w)=-\sum_{n=1}^{N}\ln\{\sum_{k=1}^{K}\pi_k(x_n,w)N(t_n\mid\mu_k(x_n,w),\sigma_k^2(x_n,w))\}$。该误差函数还涉及给定参数下高斯似然函数的计算。这样我们可以定义前向传播算法，来计算预测分布的参数与误差，具体伪代码如下：

前向传播算法

输入：样本 x；网络参数w_1、b_1、w_2、b_2

1. 经过隐含层与激活函数：$z_1=w_1x+b_1\quad a_1=\tanh(z_1)$；
2. 经过输出层得到网络净输出值：$z_2=w_2a_1+b_2$；
3. 划分输出值分别得到：$a_2^\pi=z_2[0:2]\quad a_2^\mu=z_2[3:5]\quad a_2^\sigma=z_2[6:8]$；
4. 计算各预测参数：$\pi(x)=\mathrm{softmax}(a_2^\pi)\quad \mu(x)=a_2^\mu\quad \sigma(x)=\exp(a_2^\sigma)$；
5. 根据误差函数计算误差。

输出：$\pi(x)\quad\mu(x)\quad\sigma(x)$

通过前向算法，我们已经得到预测分布以及与真实值之间的误差。接下来需要根据误差计算对各层参数的偏导数。混合系数$\pi_k(x)$看作 x 的先验概率，我们引入对应的后验概率：$\gamma_k(t\mid x)=\dfrac{\pi_k N_{nk}}{\sum_{l=1}^{K}\pi_l N_{nl}}$，其中$N_{nk}$表示$N(t_n\mid\mu_k(x_n),\ \sigma_k^2(x_n))$。通过求导，我们可以得到误差函数对各个网络的净输出的导数分别为：

$$\frac{\partial E_n}{\partial a_2^\pi}=\pi(x)-\gamma(x)\quad \frac{\partial E_n}{\partial a_2^\mu}=\gamma(x)\{\frac{\mu(x)-t_n}{\sigma^2(x)}\}\quad \frac{\partial E_n}{\partial a_2^\sigma}=-\gamma(x)\{\frac{\|\mu(x)-t_n\|^2}{\sigma^2(x)}-1\}\qquad（1）$$

我们利用反向传播算法计算误差对前面各层网络参数的梯度，并采用随机梯度下降的方法更新权重。具体伪代码如下：

反向传播算法

输入：预测分布参数： $\pi(x)\quad \mu(x)\quad \sigma(x)$；**网络参数：** w_1、b_1、w_2、b_2；

网络各层输出值： z_1、a_1、z_2；**学习率：** α

1. 根据公式（1）计算误差对网络净输出的导数，并组合得到导数：$\delta^{(2)}=\frac{\partial E_n}{\partial z_2}=\begin{bmatrix}\frac{\partial E_n}{\partial a_2^{\pi}}\\ \frac{\partial E_n}{\partial a_2^{\mu}}\\ \frac{\partial E_n}{\partial a_2^{\sigma}}\end{bmatrix}$；

2. 根据求导公式，我们可以计算得到：$\frac{\partial E_n}{\partial b_2}=\delta^{(2)}\quad \frac{\partial E_n}{\partial w_2}=\delta^{(2)}a_1^T$；

3. 根据链式求导法则，我们可以得到误差对第一层参数的导数：

$\delta^{(1)}=w_2^T\delta^{(2)}\bullet\tanh'(z_1)\quad \frac{\partial E_n}{\partial b_1}=\delta^{(1)}\quad \frac{\partial E_n}{\partial w_1}=\delta^{(1)}x^T$ 。

4. 我们利用随机梯度下降法来更新网络参数：

$$\begin{aligned}w_1&=w_1-\alpha\frac{\partial E_n}{\partial w_1}\quad b_1=b_1-\alpha\frac{\partial E_n}{\partial b_1}\\ w_2&=w_2-\alpha\frac{\partial E_n}{\partial w_2}\quad b_2=b_2-\alpha\frac{\partial E_n}{\partial b_2}\end{aligned}$$ 。

输出：网络参数： w_1、b_1、w_2、b_2

至此，我们完成了一次迭代训练网络的流程。实际中，我们需要迭代多次才能使得模型收敛。

2. 算法实现

我们将混合密度网络封装为一个类，具体实现了前向传播算法、反向传播算法以及训练、预测函数。我们依次介绍各个函数的实现。

```python
class MDN(object):
    def __init__(self, n_input=1, n_output=1,
                       n_hidden=24, n_components=3, alpha=1e-4):
        # 学习率
        self.alpha = alpha
        self.epsilon = 1e-9
        # 输入特征维数
        self.n_input = n_input
        # 输出目标维数
        self.n_output = n_output
        # 隐含层神经元数
        self.n_hidden = n_hidden
        # 输出组成成分数
        self.n_components = n_components
        # 权重
         self.w = [np.random.normal(scale=np.sqrt(0.01), size=(n_hidden, n_input)),
                   np.random.normal(scale=np.sqrt(0.01), size=((n_output+2)*n_components, n_hidden))]
        # 偏置
         self.b = [np.random.normal(scale=np.sqrt(0.01), size=(n_hidden, 1)),
                   np.random.normal(scale=np.sqrt(0.01), size=((n_output+2)*n_components, 1))]
```

混合密度网络的初始化需要包括：输入值的维数、输出值的维数、隐含层神经元数量、混合成分数量以及学习率。同时我们还需要初始化网络的权重与偏置。

```python
def __forward(self, x):
    '''''
        前向传播算法
    '''
    hiddens = []
    activations = [x]

    h = self.w[0] @ activations[-1] + self.b[0]
    hiddens.append(h)
    a = self.__tanh(h)
    activations.append(a)

    h = self.w[1] @ activations[-1] + self.b[1]
    hiddens.append(h)
```

```
        a = self.__output_activation(h)
        activations.append(a)

        return (hiddens, activations)
```

上面是前向传播算法的实现，我们传入样本值 x 后依次计算各层权重对上一层输出的线性加权，并经过相应的激活函数得到该层输出值。

```
    def __backward(self, cache, t):
        '''''
            反向传播算法
        '''
        hiddens, activations = cache
        nable_w = [np.zeros(w.shape) for w in self.w]
        nable_b = [np.zeros(b.shape) for b in self.b]

        pi, mu, sigma = activations[-1]
        N = self.__normal_likelihood(t, mu, sigma)
        loss = self.__loss_function(pi, N)
        # 计算后验概率
        gamma = (pi * N) / np.sum(pi * N, axis=0)
        # 误差函数对分布参数的偏导数计算
        gd_pi = pi - gamma
        gd_mu = gamma * (mu - t) / np.square(sigma)
        gd_sigma = - gamma * (np.square(t - mu)
                             / np.power(sigma, 2) - 1)
        # 反向求解各层梯度
        delta = np.vstack((gd_pi, gd_mu, gd_sigma))
        nable_b[-1] = np.mean(delta, axis=-1, keepdims=True)
        nable_w[-1] = delta @ activations[-2].T

        delta = self.w[-1].T @ delta * self.__tanh(hiddens[-2], flag=True)
        nable_b[-2] = np.mean(delta, axis=-1, keepdims=True)
        nable_w[-2] = delta @ activations[-3].T

        # 利用梯度更新每层参数
        self.b = [b - self.alpha*delta_b for b, delta_b in zip(self.b, nable_b)]
        self.w = [w - self.alpha*delta_w for w, delta_w in zip(self.w, nable_w)]

        return loss
```

上面是我们对反向传播算法的实现，我们传入前向过程中各层的中间结果，并利用求导公式与随机梯度下降法对各层权重进行更新。

```python
def fit(self, x, t, epoch=10, verbose=False):
    '''''
        训练函数 随机梯度下降法
    '''
    m = x.shape[1]
    for i in range(epoch):
        loss = 0
        for j in trange(m, ascii=True):
            # 选取一个样本训练
            d1 = x[:, j].reshape((-1, 1))
            d2 = t[:, j].reshape((-1, 1))
            tmp = self.__backward(self.__forward(d1), d2)
            loss += tmp
        if verbose:
            tqdm.write(f"{loss / m}")

def predict(self, x):
    '''''
        预测函数
    '''
    _, activations = self.__forward(x)
    pi, mu, _ = activations[-1]
    pred = []

    for i, idx in enumerate(np.argmax(pi, axis=0)):
        pred.append(mu[idx, i])

    return pi, mu, pred
```

上面是我们实现的训练函数与预测函数。在训练时，我们每次从训练集选择一个样本送入网络进行前后向传播算法。而在预测时，我们根据输出分布的混合系数，选择当前数据点最大的混合系数对应的分布均值作为预测值。

四、结果与分析

1. 实验内容与步骤

我们根据公式生成训练样本后，将 t 作为输入而 x 作为目标值以求解映射 $t_n \to x_n$。实验中，我们设置隐含层神经元数量为：24，混合成分数量为：3，学习率为：0.0001。其中我们采用随机梯度下降的方法，每次从训练集中选择一个样本输入网络训练。模型经过 2500 个 epoch 的训练后停止。在预测结果时，我们将(0,1)区间均匀采样的数据点输入网络，得到对应的混合分布参数。根据混合系数的大小将对应分布的均值作为该点的预测结果。

2. 实验结果及分析

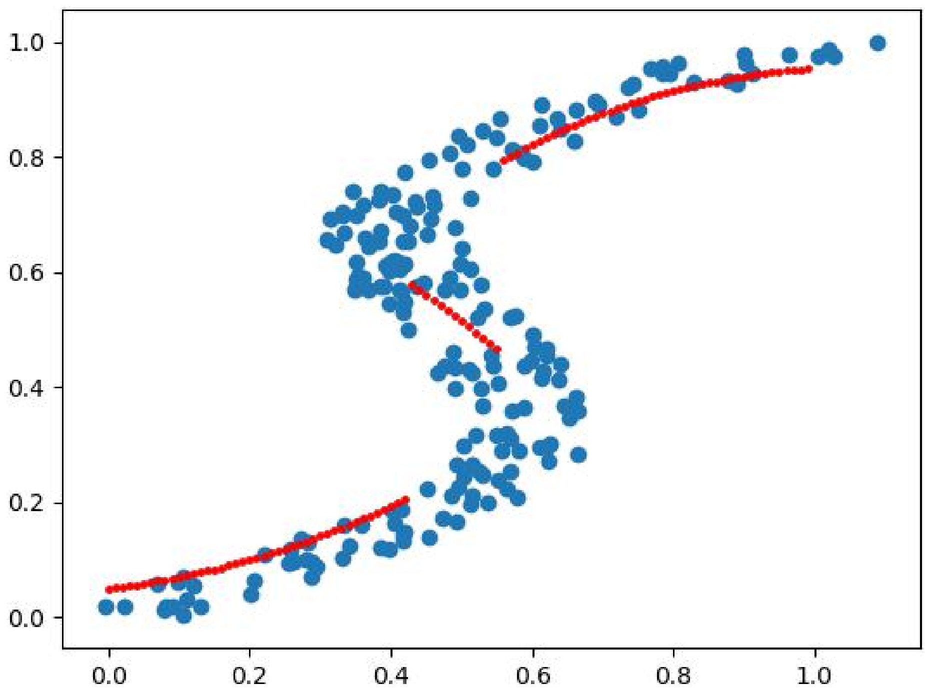

图 3 实验结果

上图为我们的模型训练结果。在本次实验中，我们的模型输出为

各个混合组成成分的参数：

$$\{\pi_1(\mathrm{x}), \mu_1(\mathrm{x}), \sigma_1(\mathrm{x})\} \quad \{\pi_2(\mathrm{x}), \mu_2(\mathrm{x}), \sigma_2(\mathrm{x})\} \quad \{\pi_3(\mathrm{x}), \mu_3(\mathrm{x}), \sigma_3(\mathrm{x})\}$$

当前输入点对应的预测分布参数如上，我们选择$\{\pi_1(x), \pi_2(x), \pi_3(x)\}$中混合系数最大值所对应的分布均值，作为条件密度函数的结果。如此一来，我们均匀采样(0,1)区间上的数据点作为输入，得到该网络的条件密度函数结果。从图 3 结果来看，我们的拟合结果还是不错的，比较能反应目标值的整体分布情况。