神经网络第二次作业

201300086 史浩男 人工智能学院

一、简述神经元联接的不同方式及作用。

1、层级结构

> 具有足够的隐藏神经元，理论上可以总是对输入和输出之间的关系建模

由两个相邻层是否完全连接，可分为全连接和局部连接

1. 前馈连接：最常见。它的信息传递只能从输入端向输出端进行，中间不允许反馈。这种连接方式适用于处理各种输入输出对之间的映射问题，例如图像识别、语音识别等。
2. 反馈连接：允许信息在神经网络中进行循环传递。信息可以从输出端返回到输入端，这种连接方式适用于处理时间序列和动态系统建模等问题。
3. 侧向连接：在神经网络中不同层之间之间连接。主要用于在同一层的神经元之间共享信息。它可以帮助网络实现不同层之间的信息共享和联合学习，从而提高网络的性能。
4. 跨层连接：将不同层神经元之间连接。它可以帮助网络实现更高级别的特征学习和表达，从而提高网络的性能。
5. 层内联接：区域内联接，加强层内神经元之间的竞争
6. 卷积连接：一种特殊的局部连接。使用一组共享的卷积核来提取下一层的特征。这种连接方式在图像、语音等领域的神经网络中广泛应用。
7. 跨通道连接：神经元之间跨越通道进行连接。可以实现跨通道信息的传递和特征学习，适用于图像、视频等领域的神经网络。

2、互联网型连接

1. Hopfield网络：一种全连接反馈型神经网络。通过学习模式之间的相似性，可以将给定的模式存储在神经元之间的权重矩阵中，并在后续的推理中使用该权重矩阵来恢复存储的模式。常用于解决模式识别、优化和最优化问题等。
2. 马尔可夫链：一种基于状态转移概率的随机过程。其中每个状态的转移概率只依赖于它的前一个状态。在神经网络中，可以将每个状态看作一个神经元，并通过定义神经元之间的转移概率来构建马尔可夫链。常用于建模具有时序关系的数据，例如语音、文本和视频等。

二、简述神经元扩展的方式及使用场景。

1. **宽度扩展**：增加神经网络中每一层的神经元数量。宽度扩展的主要优势是可以增加网络的容量，从而提高网络的**泛化能力和鲁棒性**
    - Wide ResNet是一种宽度较大的残差网络，通过增加每个残差块中的通道数来提高网络的性能。
    - 音频和语音信号处理任务，例如语音识别和语音合成等，因为它们需要对复杂的语音信号进行建模；

- 自然语言处理任务中的词嵌入和词语分类等子任务，因为它们需要对大量的词汇进行处理。

2. **深度扩展**：通过增加网络的深度来**提高网络的容量和性能，增强表征能力**，从而提高网络对数据的建模能力。
    - 在NLP中，可通过增加RNN的层数来提高对语言序列的建模能力。
    - ResNet是一种深度较大的残差网络，通过增加残差块的数量来提高网络的性能。
    - 图像和语音识别等计算机视觉和语音处理任务，因为它们需要对大量的复杂特征进行建模；

三、介绍两种不同的感知机权重初始化方法。

1、随机初始化：

将感知机的权重和偏置参数随机初始化为小的随机值，例如**服从高斯分布或均匀分布的随机数。**

优点：简单、快速，并且在某些情况下可以获得良好的表现。

缺点：由于随机初始化没有考虑网络的拓扑结构和数据的特征，因此可能会导致网络训练收敛缓慢或陷入局部最优解的问题。

2、Xavier初始化：

一种启发式的初始化方法，根据网络的输入和输出的个数自动调整权重初始化的大小。具体通过将权重初始化为均匀分布或正态分布的方式，使得**每一层的输出的方差等于其输入结点数的倒数。**

优点是可以避免梯度消失和梯度爆炸的问题，并且能够在不同的层数和激活函数下获得较好的表现。因此，Xavier初始化已成为许多深度学习框架的默认初始化方法。

四、简述 LMS 算法的优劣势，以及对应的改进措施。

Least Mean Squares算法是一种基于梯度下降的自适应滤波算法。它的主要思想是通过对误差的不断调整来迭代地更新滤波器的系数，以最小化误差的平方和。

优势：

1. 简单易部署，不依赖于模型，只需要进行简单的数学计算。
2. 自适应性好：LMS算法可以根据输入信号的特征动态地调整滤波器的系数，适应不同的信号环境。

劣势：

1. 只使用单个样本进行更新，则梯度方向不一定符合steepest descent
2. 收敛很慢，收敛性和稳定性差：受学习率的选择、信号的统计特性等影响。如果选择不当，可能会导致算法的不稳定性和收敛性差。
3. 可能出现局部最优解：由于LMS算法是基于梯度下降的优化算法，存在着收敛到局部最优解的可能性。
4. 对于高阶滤波器的应用效果差：LMS算法在处理高阶滤波器时，往往需要较长的训练时间，且容易出现过拟合的情况。

改进措施：

1. 改进学习率的选择：选择合适的学习率可以加快收敛速度和提高算法的稳定性。比如采用变化学习率的方法，可以在算法迭代初期使用较大的学习率，以快速收敛到最优解，而在后期则逐渐减小学习率，以提高算法的稳定性。
2. 采用正则化方法：为了解决LMS算法容易出现过拟合的问题，可以采用正则化方法，通过添加正则化项来限制滤波器的系数，使其更加平滑，从而提高算法的泛化性能。

五、参照 PPT 中示例一与示例二的感知机模型与代码，思考对于 Iris的完整数据集，输入和输出神经元的个数该如何设计

（文字描述即可）（注：对于完整的 Iris 数据集，输入特征有 4 个，花的种类有三种）

```python
import numpy as np

class Perceptron(object):

    def __init__(self, eta=0.01, epochs=50):
        self.eta = eta
        self.epochs = epochs

    def train(self, X, y):

        self.w_ = np.zeros(1 + X.shape[1])   # 3.初始化权重
        self.errors_ = []                    # 4.初始化误差个数

        for _ in range(self.epochs):
            errors = 0
            for xi, target in zip(X, y):     # 5.遍历所有样本
                update = self.eta * (target - self.predict(xi))   # 求解Δw
                self.w_[1:] += update * xi   # 更新权重
                self.w_[0] += update
                errors += int(update != 0.0) # 记录误差
            self.errors_.append(errors)
        return self

    def net_input(self, X):                  # 1.对输入的样本求和
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):                    # 2.使用阈值函数将求和结果映射到1或-1
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

思考题

权重更新公式

$$\Delta w_j = \eta\,(\text{target}^{(i)} - \text{output}^{(i)})\,x_j^{(i)}$$

权重更新分量公式

$$\Delta w_0 = \eta(\text{target}^{(i)} - \text{output}^{(i)})$$
$$\Delta w_1 = \eta(\text{target}^{(i)} - \text{output}^{(i)})\,x_1^{(i)}$$
$$\Delta w_2 = \eta(\text{target}^{(i)} - \text{output}^{(i)})\,x_2^{(i)}$$

权重更新：正确分类

$$\Delta w_j = \eta(-1^{(i)} - -1^{(i)})\,x_j^{(i)} = 0$$
$$\Delta w_j = \eta(1^{(i)} - 1^{(i)})\,x_j^{(i)} = 0$$

权重更新：错误分类

$$\Delta w_j = \eta(1^{(i)} - -1^{(i)})\,x_j^{(i)} = \eta(2)\,x_j^{(i)}$$
$$\Delta w_j = \eta(-1^{(i)} - 1^{(i)})\,x_j^{(i)} = \eta(-2)\,x_j^{(i}$$

思考题：权重更新有没有更简洁写法？
（提示：感知机的向量表示）

```python
import numpy as np

class AdalineGD(object):

    def __init__(self, eta=0.01, epochs=50):
        self.eta = eta
        self.epochs = epochs

    def train(self, X, y):

        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []

        for i in range(self.epochs):
            output = self.net_input(X)
            errors = (y - output)
            self.w_[1:] += self.eta * X.T.dot(errors)
            self.w_[0] += self.eta * errors.sum()
            cost = (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self

    def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        return self.net_input(X)

    def predict(self, X):
        return np.where(self.activation(X) >= 0.0, 1, -1)
```

差别在于两点

- 这里的“o”是一个实数，而不是之前所用的一个类别标签。
- 权重更新是直接作用于所有样本，而不同于之前的逐个样本更新，这种方法也叫做“批量梯度下降”（batch gradient descent）

这种“梯度下降”的思想，是一个重要的知识点，之后会详细讲解。

方案：

可以设计4个输入神经元，3个输出神经元，即每个输出神经元对应一种花的品种。

具体来说，假设输入特征为x1, x2, x3, x4，对应的权重为w1, w2, w3, w4，偏置为b，那么对于每个输出神经元i，可以计算它对应的输出值y_i：

$$y_i = f(w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4 + b_i) \tag{1}$$

其中，f是激活函数，可以选择sigmoid函数或者ReLU函数等。对于三个输出神经元，分别对应三种花的品种，可以根据输出值的大小作为概率，来确定具体的分类结果，例如可以选择输出值最大的那个神经元对应的花的品种作为分类结果。