# 机器学习

## 作业三

## 一. (20 points) 激活函数比较

神经网络隐层使用的激活函数一般与输出层不同. 请画出以下几种常见的隐层激活函数的示意图并计算其导数（导数不存在时可指明）, 讨论其优劣（每小题 4 points）:

1. Sigmoid 函数, 定义如下

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

2. 双曲正切函数 (Hyperbolic Tangent Function, Tanh), 定义如下

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2}$$

3. 修正线性函数 (Rectified Linear Unit, ReLU) 是近年来最为常用的隐层激活函数之一, 其定义如下

$$f(x) = \begin{cases} 0, & \text{if } x < 0; \\ x, & \text{otherwise.} \end{cases} \tag{3}$$

4. 指数线性单元 (Exponential Linear Unit, ELU) ， 其中 $\alpha > 0$

$$f(x) = \begin{cases} x, & \text{if } x > 0; \\ \alpha(e^{x} - 1), & \text{otherwise.} \end{cases} \tag{4}$$

5. Swish 函数, 定义如下

$$f(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}. \tag{5}$$

# 二. (30 points) 神经网络实战

请实现一个简单的全连接神经网络，并参考教材图 5.8 实现反向传播算法训练该网络，用于解决二分类问题。实验数据采用 `scikit-learn` 提供的 `make_moons` 数据集，moons 是一个简单的二分类数据集。

1. 使用 NumPy 手动实现神经网络和反向传播算法。(15 points)
2. 实现并比较不同的权重初始化方法。(5 points)
3. 在 moons 数据集上训练网络，观察并分析收敛情况和训练过程。(10 points)

**提示:**

1. 神经网络实现：
   - 实现一个具有一个隐藏层的全连接神经网络。
   - 网络结构：输入层 (2 节点) → 隐藏层 (8 节点) → 输出层 (1 节点)
   - 隐藏层使用 ReLU 激活函数，输出层使用 Sigmoid 激活函数。
   - 使用交叉熵损失函数。
2. 权重初始化方法。实现以下三种初始化方法，并比较它们的性能：
   - 正态初始化：从正态分布 $N(0,1)$ 中采样。
   - Xavier 初始化：根据前一层的节点数进行缩放。

     $$W_{ij} \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$$

     其中 $n_{in}$ 是前一层的节点数，$n_{out}$ 是当前层的节点数。
   - He 初始化：He 初始化假设每一层都是线性的，并且考虑了 ReLU 激活函数的特性。

     $$W_{ij} \sim N\left(0, \frac{2}{n_{in}}\right)$$

     其中 $n_{in}$ 是前一层的节点数。
3. 训练和分析：
   - 使用提供的 moons 数据集。
   - 实现小批量梯度下降算法。

- 训练 200 个 epoch，记录并绘制训练过程中的损失值和准确率，训练结束后绘制决策边界。
- 比较不同初始化方法对训练过程和最终性能的影响并给出合理解释。
- 尝试不同的学习率，观察其对训练的影响并给出合理解释。

```
"""
注意:
1. 这个框架提供了基本的结构，您需要完成所有标记为 'pass' 的函数。
2. 确保正确实现前向传播、反向传播和梯度更新。
3. 在比较不同初始化方法时，保持其他超参数不变。
4. 记得处理数值稳定性问题，例如在计算对数时避免除以零。
5. 尝试使用不同的学习率（例如 0.01, 0.1, 1），并比较结果。
6. 在报告中详细讨论您的观察结果和任何有趣的发现。
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

# 生成数据集
X, y = make_moons(n_samples=1000, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, init_method='random'):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # 初始化权重和偏置
        if init_method == 'normal':
            # 实现正态初始化
            pass
        elif init_method == 'xavier':
            # 实现Xavier初始化
            pass
        elif init_method == 'he':
            # 实现He初始化
            pass
        else:
            raise ValueError("Unsupported initialization method")

    def relu(self, x):
        return np.maximum(0, x)

```

```
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def forward(self, X):
        # 实现前向传播
        pass

    def backward(self, X, y, y_pred):
        # 实现反向传播
        pass

    def train(self, X, y, learning_rate, epochs, batch_size):
        # 实现小批量梯度下降训练
        pass

# 辅助函数
def plot_decision_boundary(model, X, y):
    # 绘制决策边界
    pass

def plot_training_process(losses, accuracies):
    # 绘制训练过程
    pass

# 主函数
def main():
    # 创建并训练模型
    # 绘制结果
    pass

if __name__ == "__main__":
    main()
```

Listing 1: 代码实现模板

# 三. (15 points) 软间隔 SVM

教材 6.4 节介绍了软间隔概念，用来解决线性不可分情况下的 SVM 问题，同时也可缓解 SVM 训练的过拟合问题。定义松弛变量 $\boldsymbol{\xi} = \{\xi_i\}_{i=1}^m$，其中 $\xi_i > 0$ 表示样本 $\boldsymbol{x}_i$ 对应的间隔约束不满足的程度。软间隔 SVM 问题可以表示为：

$$\max_{\boldsymbol{w},b} \rho$$

$$\text{s.t. } \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|_2} \geq \rho, \quad \forall i \in [m].$$

该式显式地表示了分类器的间隔 $\rho$。基于这种约束形式的表示，可以定义两种形式的软间隔。

- 第一种是绝对软间隔：

$$\frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|_2} \geq \rho - \xi_i.$$

- 第二种是相对软间隔：

$$\frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|_2} \geq \rho(1 - \xi_i).$$

这两种软间隔分别使用 $\xi_i$ 和 $\rho\xi_i$ 衡量错分样本在间隔上的违背程度。在优化问题中加入惩罚项 $C\sum_{i=0}^m \xi_i^p$ （其中 $C > 0, p \geq 1$），使得不满足约束的样本数量尽量小（即让 $\xi_i \to 0$）。

## 问题：

1. **(5points)** 软间隔 SVM 通常采用相对软间隔，写出其原问题的形式（要求不包含 $\rho$）。
2. **(5points)** 写出采用绝对软间隔的 SVM 原问题（不包含 $\rho$），并说明为什么一般不使用绝对软间隔来构建 SVM 问题。
3. **(5points)** 写出 $p = 1$ 情况下软间隔 SVM 的对偶问题。

# 四. (35 points) SVM 编程

在本题中，你将使用支持向量机（SVM）对一个非线性可分的数据集进行分类，并进一步与其他典型分类模型（BP 神经网络与 C4.5 决策树）进行对比分析。实验数据采用 `scikit-learn` 提供的 `make_moons` 数据集。

1. 在提供的 `make_moons` 数据集上训练一个非线性核（如 RBF 核）的支持向量机模型，并绘制分类边界。(5 points)
2. 调整正则化系数 $C$ 与 RBF 核宽度 $\gamma$，观察分类边界的变化情况，并分析参数对模型复杂度与过拟合的影响。(5 points)
3. 比较 RBF 核与线性核 SVM 的分类效果，简要说明两者在处理非线性数据时的差异原因。(5 points)
4. 在训练集上随机翻转一部分标签（例如 10%），比较 SVM 的性能变化，并分析 SVM 对噪声敏感的原因。(5 points)
5. 进一步在相同数据分布下分别使用原始数据集和加噪数据集进行训练与测试。:
   - 一个基于反向传播算法的多层感知机（BP 神经网络）;
   - 一个基于信息增益率划分的 C4.5 决策树模型。

   对三种模型（SVM、BP 神经网络、C4.5 决策树）进行分类结果可视化与性能比较（准确率、精确率、召回率、F1 分数）。(10 points)
6. 根据实验结果，从模型结构、非线性拟合能力、泛化性能和对噪声敏感度等角度，对三者的优劣进行综合分析。(5 points)

**提示:**

- 使用 `SVC` 类(支持向量分类器)实现 SVM 模型,可选择不同核函数(`kernel='rbf'` 或 `kernel='linear'`);
- 输出训练集与测试集的准确率（可用 `accuracy_score`）;
- 可使用 `matplotlib` 的 `contourf()` 与 `scatter()` 函数绘制分类边界与样本点;
- 调整参数 `C`（正则化系数）与 `gamma`（RBF 核宽度）观察分类边界变化。
- 在实现部分，可使用 `MLPClassifier`（设置隐藏层规模为 $(10, 10)$）构建 BP 神经网络，使用 `DecisionTreeClassifier` 构建 C4.5 决策树。
- 为了方便对比，可以将三种模型的性能结果整理成表格，并在图中展示三者的分类边界差异。

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

 # 生 成 数 据 集
X, y = make_moons(n_samples=1000, noise=0.1, random_state=42)
X_train , X_test, y_train , y_test = train_test_split(X, y, test_size=0.2, random_state
    =42)

def train_and_plot_svm(X_train, X_test, y_train, y_test, kernel, C, gamma):
    """训练SVM并绘制分类结果"""
    pass

def train_and_plot_bp(X_train, X_test, y_train, y_test):
    """训练BP神经网络并绘制分类结果"""
    pass

def train_and_plot_c45(X_train, X_test, y_train, y_test):
    """训练C4.5 决策树并绘制分类结果"""
    pass

def main():
    # 训练 SVM、BP神经网络、C4.5决策树
    pass

if __name__ == "__main__":
    main()
```

Listing 2: 代码实现模板