HW-3-陈宸

Q1. 使用Pytorch实现MLP 对Circle分类

补全mlp层结构如下：

```
class MLPModel(torch.nn.Module):
    def __init__(self, input_dim, hidden_size, output_dim):
        super(MLPModel, self).__init__()
        # TODO: implement your 2-layer MLP here
        self.mlp_1 = torch.nn.Linear(input_dim, hidden_size)
        self.to_outputLayer = torch.nn.Linear(hidden_size, output_dim)

    def forward(self, x):
        # TODO: Implement forward function here
        x = self.mlp_1(x)
        x = torch.sigmoid(x)
        outputs = torch.sigmoid(self.to_outputLayer(x))
        return outputs
```

不同参数下达到90%准确率的迭代次数如下：

```
# lr = 0.1,hidden_size = 3,epoch = 19380
Test - Loss: 0.18537183105945587. Accuracy: 96.36363636363636
Train -  Loss: 0.1958434134721756. Accuracy: 95.07462686567165

# lr = 0.2,hidden_size = 3,epoch = 13985
Test - Loss: 0.3154851794242859. Accuracy: 88.48484848484848
Train -  Loss: 0.3127270042896271. Accuracy: 90.44776119402985

# lr = 0.3,hidden_size = 3,epoch = 5410
Test - Loss: 0.2835902273654938. Accuracy: 91.81818181818181
Train -  Loss: 0.27574992179870605. Accuracy: 91.7910447761194

# lr = 0.4,hidden_size = 3,epoch = 3619
Test - Loss: 0.09714648872613907. Accuracy: 96.36363636363636
Train -  Loss: 0.12452100217342377. Accuracy: 96.11940298507463
```

观察不同lr和hidden_size取值下的的精度，得出：**可行的最小隐藏层单元数为3。**

取四个象限(0.5,0.5),(2,2),(-0.5,0.5),(-2,2),(0.5,-0.5),(2,-2),(-0.5,-0.5),(-2,-2)的八个点分别对应y=0和y=1。

将八个x1、x2代入lr=0.3,hidden_size=3下模型的参数，手动利用python进行模拟前向传播过程。

```
# calculate_output.py
import numpy as np

# hidden_size: 3
# mlp_1.weight
```

```
# tensor([[3.9119, -1.6289],
#         [-3.2995, -2.6610],
#         [0.6128, -3.9641]])
# mlp_1.bias
# tensor([-2.6002, -2.7359, 2.6362])
# to_outputLayer.weight
# tensor([[-18.4417, -18.6520, 18.7927]])
# to_outputLayer.bias
# tensor([-4.4406])


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


def neural_network_output(x1, x2):
    x = np.array([x1, x2])
    W1 = np.array([[3.9119, -1.6289],
                   [-3.2995, -2.6610],
                   [0.6128, -3.9641]])
    b1 = np.array([-2.6002, -2.7359, 2.6362])
    W2 = np.array([[-18.4417, -18.6520, 18.7927]])
    b2 = np.array([-4.4406])

    h1 = sigmoid(np.dot(W1, x) + b1)
    y = sigmoid(np.dot(W2, h1) + b2)
    return y


# output
points = [(0.5, 0.5), (2, 2), (-0.5, 0.5), (-2, 2), (-0.5, -0.5), (-2, -2), (0.5, -0.5), (2,
-2)]
outputs = [float(neural_network_output(x1, x2)) for x1, x2 in points]
print(outputs)
```

手动计算分类结果如下：

[0.9963535445012736, 1.526549576777541e-09, 0.9930364058834007, 0.0003591781040789834, 0.9614500530789392, 0.013204950894971125, 0.966870367602409, 0.011924361477611355]

近似输出值大于0.5则分类为在圆内，反之分类为在圆外。根据结果，发现所有点的分类结果均正确。

Q2. 使用Pytorch实现MLP 对Quarter分类

修改use_circle_quarter=1，调节参数，发现lr=0.1，hidden_size=1时经过500次迭代准确率已达到90%以上；hidden_size=2时，准确率于迭代过程的70%后有所下降，降至90%以下；hidden_size = 3后才能满足随hidden_size增加，迭代过程中网络的分类准确率稳定在90%以上。

lr = 0.2及以上时，令hidden=1准确率即可达到90%以上。**相较于Q1，为获取较高准确率所需的隐藏层数目降低。最小可行值为1。**

取(0.5,0.5)和(2,2)，手动利用python代码计算输出结果如下：

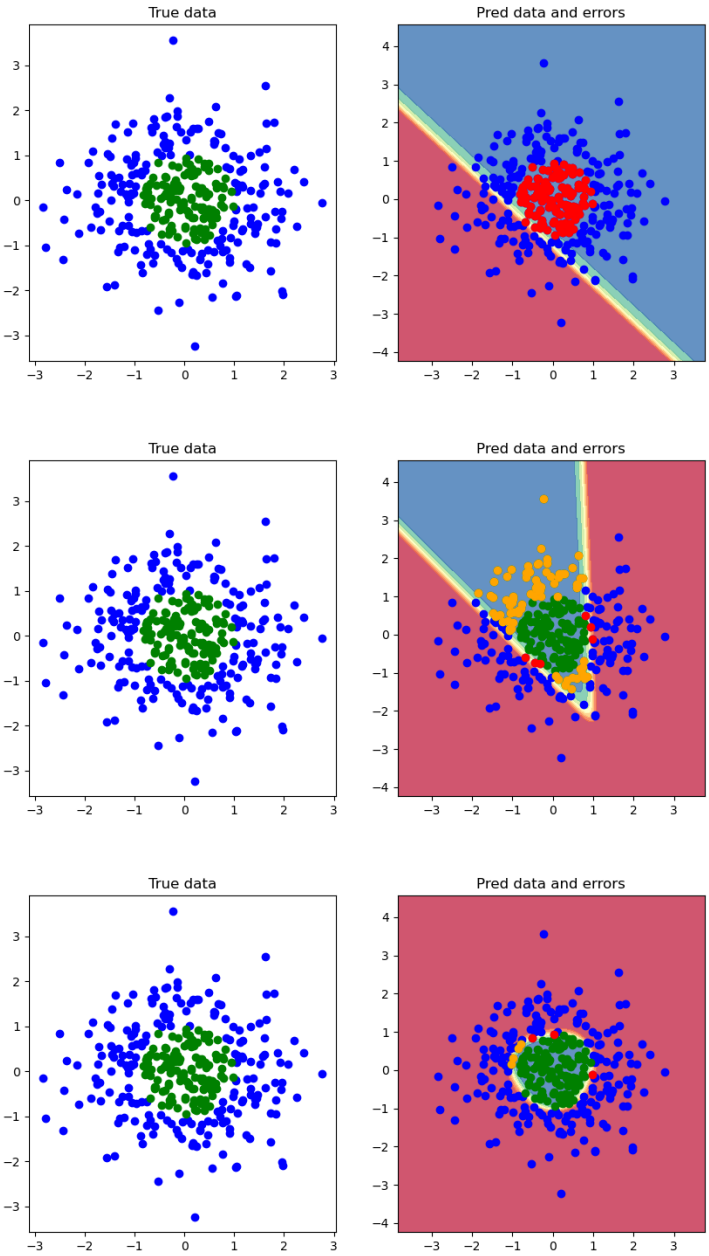
[0.7189702539083668, 0.0015011658710572728]，点(0.5,0.5)的输出近似值大于0.5；点(2,2)输出近似值小于0.5，分类结果正确且较好。

Q3. 可视化分类平面

修改vis_prediction函数，传入model作为参数以便可视化模型的分类边界。利用meshgrid函数创建数据点网格后使用pyplot.contourf()绘制分类平面。

```
def vis_prediction(X_test, y_test, y_pred, model):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

    ax1.scatter(X_test[y_test <= 0.5, 0], X_test[y_test <= 0.5, 1], c='b')
    ax1.scatter(X_test[y_test > 0.5, 0], X_test[y_test > 0.5, 1], c='g')
    ax1.set_title('True data')

    # Create a meshgrid
    x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
    y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

    # Predict on the meshgrid
    grid = torch.Tensor(np.c_[xx.ravel(), yy.ravel()])
    with torch.no_grad():
        zz = model(grid).reshape(xx.shape)
    zz = zz.numpy()

    x0_tensor = X_test[y_pred <= 0.5, 0]
    y0_tensor = X_test[y_pred <= 0.5, 1]
    x1_tensor = X_test[y_pred > 0.5, 0]
    y1_tensor = X_test[y_pred > 0.5, 1]
    # Plot the decision boundary
    ax2.contourf(xx, yy, zz, alpha=0.8, cmap=plt.cm.Spectral)
    error_1 = np.logical_and(y_pred <= 0.5, y_test > 0.5)
    error_2 = np.logical_and(y_pred > 0.5, y_test <= 0.5)
    ax2.scatter(x0_tensor, y0_tensor, c='b')
    ax2.scatter(x1_tensor, y1_tensor, c='g')
    ax2.scatter(X_test[error_1, 0], X_test[error_1, 1], c='r')
    ax2.scatter(X_test[error_2, 0], X_test[error_2, 1], c='orange')
    ax2.set_title('Pred data and errors')

    plt.show()
```

hidden_size分别为1,2,3时的分类边界可视化如下，蓝色区域分类标签为1，红色区域分类标签为0。



选取了$x^2+y^2=1$圆周上点a（$\frac{\sqrt{2}}{2}$, $\frac{\sqrt{2}}{2}$)和b（0.75, 0.75)作为特例数据。根据输出的模型参数，调整调用calculate_output函数。输出结果为[0.7509926743824804, 0.4910385710773224]，分类标签分别为1和0，但实际上点a在圆周上而分类器将其分类为在圆内；点b在圆周外，分类器将其正确分类为在圆外，但十分接近于被分类为在圆内的错误结果，说明：在hidden_size较小的条件下，该mlp对边界处各点的分类准确率仍不够理想，若能提高hidden_size以建立更多的分类平面，或者增加隐藏层数目，则可使多个线性的分类平面综合作用的分类结果近似等同于非线性分类平面。