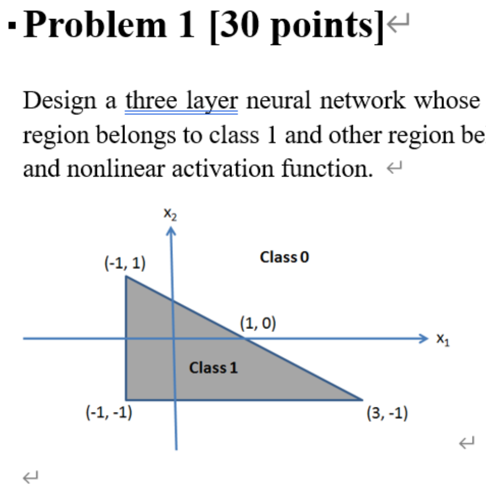
Student’s ID: 22307110187
Name: 谢志康
PART2 Solution:



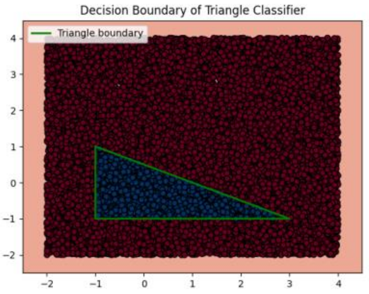
▪**Problem 1 [30 points]**↵

Design a three layer neural network whose decision boundary is as shown in Figure 1. The gray region belongs to class 1 and other region belongs to class 0. Show your network structure, weights and nonlinear activation function. ↵

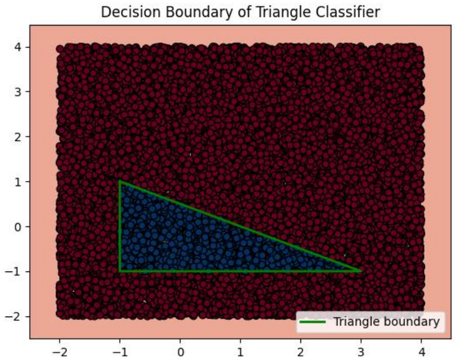
二分类问题 三角形区域灰色，分为 class1，其余部分分为 class0
可以直接使用线性边界来分割，像线性规划那样，三条直线，相交为一个三角形
Line1: from （-1, 1） to （3, -1）　　x1 + 2 * x2 = 1
Line2: from （-1, 1） to （-1, -1）　　x1 = -1
Line3: from （-1, -1） to （3, -1）　　x2 = -1

三层 neural network：
线性输入层，输入特征，二维，两个神经元（x1, x2）
隐藏层，进行线性分类，搞多个神经元，分类边界
输出层，使用 sigmoid 激活函数，二分类

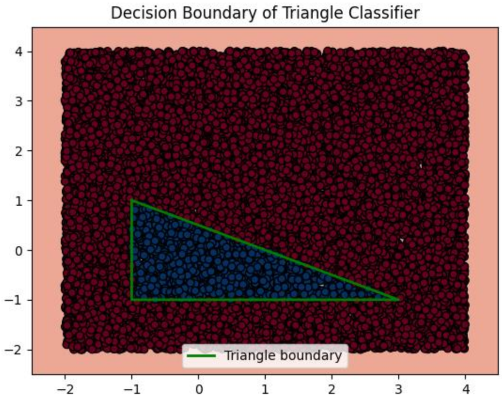
整体网络结构，输入层一定是 2 个神经元，隐藏层试了一下用 16 个神经元，不会跑太慢，精确度也还行，输出层 1 个神经元，后接 sigmoid 激活函数，做二分类。
一到二，二到三层都用 relu 激活函数，最后用 sigmoid 激活函数。
随机生成 15000 个点（pixel 选小一点，15000 个点基本能占据全屏）
使用上文我们定义的 model 开始训练，loss 计算使用二元交叉熵损失函数，优化器使用随机梯度下降法。
Experiment：蓝色表示分类在 class1（即题目灰色部分）的点，红色表示分类在 class0 的点，绿色边框表示正确边界。
Epoch：2000　LR：0.01　　结果：Loss：0.2052



Epoch：2000 LR：0.1   结果：Loss：0.0736



Learning rate 往大调比较好
Epoch：10000 LR：0.8   结果：Loss：0.0074 几乎已经没有 loss



最终每层变换的 weights 和 bias 输出如下：

Layer 1 Weights (Input -> Hidden Layer): 2 维转为 16 维

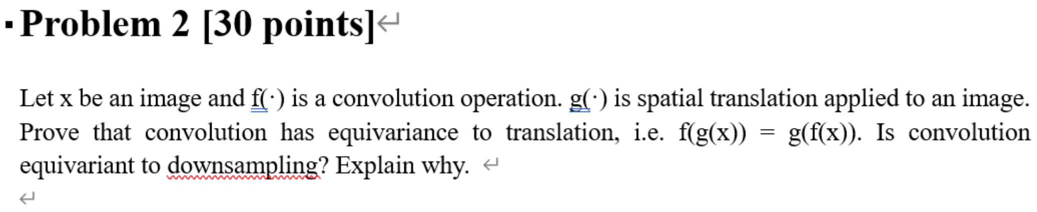
```
tensor([[ 5.7999e-02, -4.9546e-01],
        [-5.8495e-02, -4.1063e+00],
        [ 2.7774e-04, -2.7898e+00],
        [-1.3166e+00,  3.4334e-02],
        [-1.2361e+00, -1.6019e-01],
        [ 1.4867e+00,  3.0269e+00],
        [-1.9841e+00,  5.0543e-02],
        [ 7.4672e-01,  1.4675e+00],
        [ 1.8198e+00,  3.5770e+00],
        [ 6.5103e-04, -3.7858e+00],
        [-4.4854e+00,  4.2988e-02],
        [ 3.5645e-04, -2.7698e+00],
        [ 2.8183e+00,  5.7381e+00],
        [-2.6319e+00, -9.9854e-02],
        [-5.6253e-01, -1.9057e-02],
        [ 4.2552e-01,  3.8779e-01]])
Layer 1 Biases:
tensor([-0.4204, -3.7348, -2.5401, -1.1699, -1.0112, -1.2652, -1.7620, -0.6202,
        -1.5120, -3.4477, -3.9744, -2.5221, -2.3983, -2.2514, -0.4810, -0.0089])

Layer 2 Weights (Hidden Layer -> Output Layer): 16维转为1维
tensor([[-0.5427, -5.4853, -3.6631, -1.6332, -1.3411, -3.5848, -2.6337, -1.5408,
         -4.2796,  -5.0988,  -5.9193,  -3.6785,  -6.7943,  -3.3752,  -0.6776,  -
0.4170]])
Layer 2 Biases:
tensor([5.6056])
```

全代码如下：（python）

```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

class TriangleClassifier(nn.Module):
    def __init__(self):
        super(TriangleClassifier, self).__init__()
        self.fc1 = nn.Linear(2, 16)
        self.fc2 = nn.Linear(16, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(x)
        x = self.sigmoid(self.fc2(x))
        return x

def is_point_in_triangle(x, y):
    flag1 = x + 2*y <= 1
    flag2 = x >= -1
    flag3 = y >= -1
    return flag1 and flag2 and flag3

def generate_data(n_samples=15000):
    X = np.random.uniform(-2, 4, (n_samples, 2))
    y = np.array([1 if is_point_in_triangle(x[0], x[1]) else 0 for x in X])
    return X, y

X, y = generate_data(15000)
X_train = torch.FloatTensor(X)
y_train = torch.FloatTensor(y).view(-1, 1)
```

```
model = TriangleClassifier()
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.8)
epochs = 10000

for epoch in range(epochs):
    model.train()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 100 == 0:
        print(f’Epoch [{epoch + 1}/{epochs}], Loss: {loss.item():.4f}’)

# Visualize decision boundary
def plot_decision_boundary(model, X, y):
    # Define the range for the plot
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max,
0.01))
    grid = np.c_[xx.ravel(), yy.ravel()]

    # Predict the probability for each point in the grid
    with torch.no_grad():
        Z = model(torch.FloatTensor(grid))
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], cmap=″RdBu″, alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=″RdBu″, edgecolors=’k’)

    # Plot the triangle boundary
    triangle_points = np.array([[-1, 1], [-1, -1], [3, -1], [-1, 1]])  # Triangle
vertices with a closing point
    plt.plot(triangle_points[:, 0], triangle_points[:, 1], ’g-’, linewidth=2,
label=″Triangle boundary″)

    plt.title(″Decision Boundary of Triangle Classifier″)
    plt.legend()
    plt.show()
```

```
plot_decision_boundary(model, X, y)

def print_model_weights(model):
    print("Layer 1 Weights (Input -> Hidden Layer):")
    print(model.fc1.weight.data)
    print("Layer 1 Biases:")
    print(model.fc1.bias.data)
    print("\nLayer 2 Weights (Hidden Layer -> Output Layer):")
    print(model.fc2.weight.data)
    print("Layer 2 Biases:")
    print(model.fc2.bias.data)


print_model_weights(model)
```



Problem 2 [30 points]

Let x be an image and f(·) is a convolution operation. g(·) is spatial translation applied to an image. Prove that convolution has equivariance to translation, i.e. f(g(x)) = g(f(x)). Is convolution equivariant to downsampling? Explain why.

Function f is a convolution operation （卷积）

Function g is a spatial translation （空间平移）

To prove: f(g(x)) == g(f(x))

也就是证明，对图像先平移后卷积和先卷积再平移得到的结果相同。

对一个图像任意坐标（像素点）对任意一个图像 x （i, j） 平移也就是线性加向量，不妨设平移后该点位置由 i,j 变为 i-t1, j-t2， 即

$$g(x)(i,j) = x(i - t_1, j - t_2)$$

图像卷积公式：

$$f(x)(i,j) = (k * x)(i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} k(m,n)x(i-m, j-n)$$

因此，当先进行 g 再进行 f 时：

X(i, j) → x(i-t1,j-t2)

再卷积变为：

$$f(g(x))(i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} k(m,n)x(i-m-t_1, j-n-t_2)$$

当先 f 再 g 时：

$$f(x)(i,j) = (k*x)(i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} k(m,n)x(i-m, j-n)$$

$$g(f(x))(i,j) = f(x)(i-t_1, j-t_2) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} k(m,n)x((i-t_1)-m, (j-t_2)-n)$$

也就是所有的 i 变成 i-t1，所有的 j 变成 j-t2，且卷积后是累加的式子，直接进行参数变换没有任何影响。

所以 f(g(x)) == g(f(x)) 证毕

卷积与下采样 equivariant 吗？

下采样也就是把 x(i,j) 变为 x(si,sj) 其中 s 是 downsampling factor

令 d(x)表示这一操作过程 即 d(x) (i,j) == x(si, sj)

而图像卷积公式如下：

$$f(x)(i,j) = (k*x)(i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} k(m,n)x(i-m, j-n)$$

F(d(x)) 也就是将所有 i 换成 si， 所有 j 换成 sj，如上公式其它不变，最后一项变成

X(si-m, sj-n)

D(f(x)) 也就是将图像的所有像素点横纵坐标伸缩 s 倍，即上式最后一项变为

X(s(i-m), s(j-n))

不同，即 fd(x) != df(x) ，卷积与下采样不是 equivariant 的。