Homework 7

Problem 1

A(5, 8). B(6, 4). C(3, 5).

1.

Substitute the coordinates into the formula and we get:

$L_1(AC)$: $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$. $-1 - 3x_1 + 2x_2 = 0$

$L_2(AB)$: $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{28} \end{bmatrix}$. $-1 + \frac{1}{2}x_1 + \frac{1}{28}x_2 = 0$

$L_3(BC)$: $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{18} \\ \frac{1}{6} \end{bmatrix}$. $-1 + \frac{1}{18}x_1 + \frac{1}{6}x_2 = 0$

2.

Substitute coordinate B into L_1, we get B lies on the area $L_1 < 0$.

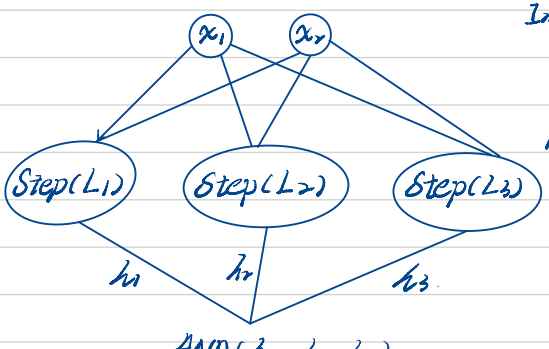
C into L_2, C lies on the area $L_2 < 0$.

A into L_3, A lies on the area $L_3 > 0$.

Therefore the 3 half planes are $\begin{cases} L_1 < 0 \\ L_2 < 0 \\ L_3 > 0 \end{cases}$

3.

Input:



Input:

coordinate (x_1, x_2)

Hidden layer:

Hidden Layer:

$\begin{cases} h_1 = \text{Step}(-L_1) \\ h_2 = \text{Step}(-L_2) \\ h_3 = \text{Step}(L_3) \end{cases}$

Output

Output:

Output = $h_1 \times h_2 \times h_3$;

Problem 2

Develop a MLP to classify the quality of white wine using the dataset winequality-white.csv. It has 7 class labels (last column): 3, 4, 5, 6, 7, 8, 9. You should convert the labels into one-hot encoding. Also, you should perform feature normalization before applying them to the MLP. Partition the data (Holdout, stratified) into 60/20/20 partitions where 60% are for training, 20% are for validation and the remaining 20% are for testing (and should never be used during training)

In [15]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt

# makes printing more human-friendly
np.set_printoptions(precision=3, suppress=True)
```

In [16]:
```
# Import data
data = pd.read_csv('winequality-white-2.csv')

X = data.drop(columns=['quality']).values
y = data['quality'].values

y = y - 3  # Shift labels to be in range [0, 6]

# Feature standardization (Normalization)
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Split data (60/20/20)
X_train, X_temp, y_train, y_temp = train_test_split(X_normalized, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Convert to PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

# Create DataLoader for batch processing
batch_size = 32
train_dataset = TensorDataset(X_train, y_train)
val_dataset = TensorDataset(X_val, y_val)
test_dataset = TensorDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

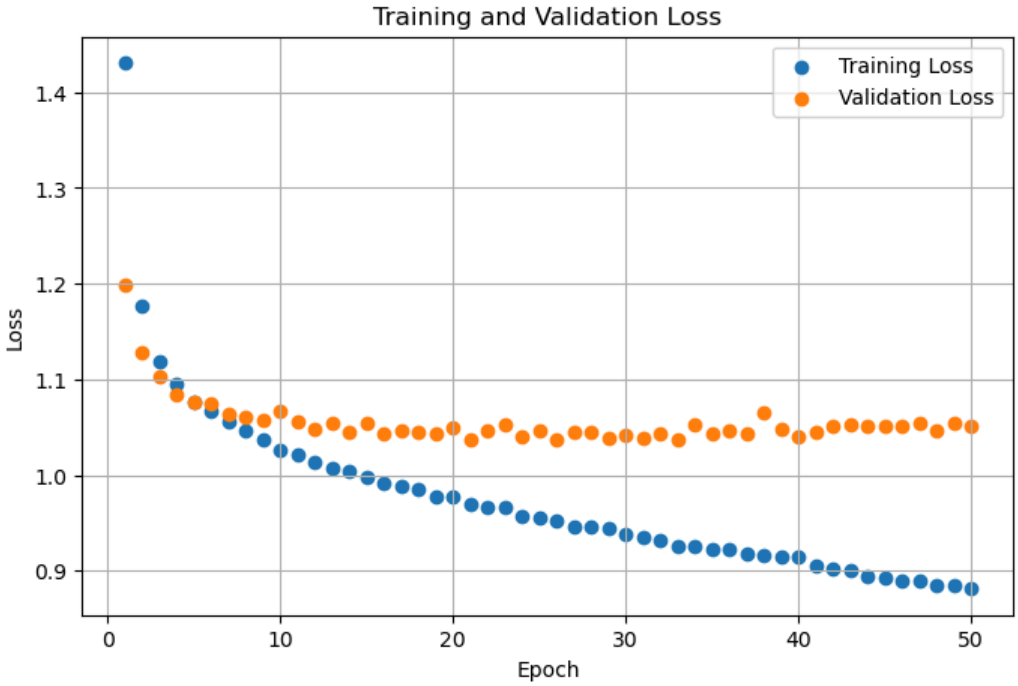
In [17]:
```
# Define the MLP model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, num_classes):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, num_classes)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        x = self.relu2(x)
        x = self.fc3(x)
        return x

# Initialize model, loss function, and optimizer
input_size = X_train.shape[1]
```

```
hidden_size1 = 64
hidden_size2 = 32
num_classes = 7

model = MLP(input_size, hidden_size1, hidden_size2, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [18]:

```
# Training
num_epochs = 50
train_losses, val_losses = [], []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * X_batch.size(0)
    train_loss = running_loss / len(train_loader.dataset)
    train_losses.append(train_loss)

    # Evaluate on validation set
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            outputs = model(X_batch)
            loss = criterion(outputs, y_batch)
            val_loss += loss.item() * X_batch.size(0)
    val_loss /= len(val_loader.dataset)
    val_losses.append(val_loss)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}')

# Evaluate the model on the test set
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        outputs = model(X_batch)
        _, predicted = torch.max(outputs, 1)
        total += y_batch.size(0)
        correct += (predicted == y_batch).sum().item()

test_accuracy = correct / total
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

# Visualize the training and validation losses
plt.figure(figsize=(8, 5))
plt.scatter(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.scatter(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid()
plt.show()
```

```
Epoch [1/50], Train Loss: 1.4306, Val Loss: 1.1981
Epoch [2/50], Train Loss: 1.1761, Val Loss: 1.1273
Epoch [3/50], Train Loss: 1.1191, Val Loss: 1.1031
Epoch [4/50], Train Loss: 1.0943, Val Loss: 1.0835
Epoch [5/50], Train Loss: 1.0760, Val Loss: 1.0765
Epoch [6/50], Train Loss: 1.0662, Val Loss: 1.0747
Epoch [7/50], Train Loss: 1.0558, Val Loss: 1.0638
Epoch [8/50], Train Loss: 1.0463, Val Loss: 1.0612
Epoch [9/50], Train Loss: 1.0372, Val Loss: 1.0572
Epoch [10/50], Train Loss: 1.0261, Val Loss: 1.0671
Epoch [11/50], Train Loss: 1.0220, Val Loss: 1.0555
Epoch [12/50], Train Loss: 1.0129, Val Loss: 1.0479
Epoch [13/50], Train Loss: 1.0077, Val Loss: 1.0539
Epoch [14/50], Train Loss: 1.0041, Val Loss: 1.0448
Epoch [15/50], Train Loss: 0.9982, Val Loss: 1.0548
Epoch [16/50], Train Loss: 0.9920, Val Loss: 1.0430
Epoch [17/50], Train Loss: 0.9876, Val Loss: 1.0471
Epoch [18/50], Train Loss: 0.9845, Val Loss: 1.0451
Epoch [19/50], Train Loss: 0.9771, Val Loss: 1.0438
Epoch [20/50], Train Loss: 0.9775, Val Loss: 1.0489
Epoch [21/50], Train Loss: 0.9695, Val Loss: 1.0364
Epoch [22/50], Train Loss: 0.9656, Val Loss: 1.0467
Epoch [23/50], Train Loss: 0.9670, Val Loss: 1.0531
Epoch [24/50], Train Loss: 0.9570, Val Loss: 1.0402
Epoch [25/50], Train Loss: 0.9560, Val Loss: 1.0456
Epoch [26/50], Train Loss: 0.9518, Val Loss: 1.0367
Epoch [27/50], Train Loss: 0.9455, Val Loss: 1.0453
Epoch [28/50], Train Loss: 0.9462, Val Loss: 1.0453
Epoch [29/50], Train Loss: 0.9442, Val Loss: 1.0382
Epoch [30/50], Train Loss: 0.9373, Val Loss: 1.0419
Epoch [31/50], Train Loss: 0.9347, Val Loss: 1.0380
Epoch [32/50], Train Loss: 0.9310, Val Loss: 1.0435
Epoch [33/50], Train Loss: 0.9254, Val Loss: 1.0372
Epoch [34/50], Train Loss: 0.9254, Val Loss: 1.0526
Epoch [35/50], Train Loss: 0.9222, Val Loss: 1.0426
Epoch [36/50], Train Loss: 0.9216, Val Loss: 1.0471
Epoch [37/50], Train Loss: 0.9170, Val Loss: 1.0437
Epoch [38/50], Train Loss: 0.9157, Val Loss: 1.0650
Epoch [39/50], Train Loss: 0.9148, Val Loss: 1.0479
Epoch [40/50], Train Loss: 0.9142, Val Loss: 1.0401
Epoch [41/50], Train Loss: 0.9046, Val Loss: 1.0453
Epoch [42/50], Train Loss: 0.9014, Val Loss: 1.0506
Epoch [43/50], Train Loss: 0.9006, Val Loss: 1.0528
Epoch [44/50], Train Loss: 0.8944, Val Loss: 1.0509
Epoch [45/50], Train Loss: 0.8929, Val Loss: 1.0507
Epoch [46/50], Train Loss: 0.8897, Val Loss: 1.0508
Epoch [47/50], Train Loss: 0.8891, Val Loss: 1.0536
Epoch [48/50], Train Loss: 0.8840, Val Loss: 1.0456
Epoch [49/50], Train Loss: 0.8842, Val Loss: 1.0548
Epoch [50/50], Train Loss: 0.8812, Val Loss: 1.0509
Test Accuracy: 53.37%
```



Problem 3

1. AND operator

To implement AND operator. let $w_i = 1$, $i = 1, 2, \dots, n$, we have $\begin{cases} w_0 + n > 0 \\ w_0 + n - 1 < 0 \end{cases}$

Therefore $-n < w_0 < -n + 1$

we have $w_0 = -n + 0.5$. $w_i = 1$, $(i = 1, 2, \dots, n)$

$w = [-n + 0.5 \quad 1 \quad 1 \quad \dots \quad 1]_{n+1}$

Only when every $x_i = 1$. $y = Step(Xw) = step(-n + 0.5 + n) = step(0.5) = 1$

even only one $x_i = 0$. $y = Step(Xw) = step(-n + 0.5 + n - 1) = Step(-0.5) = 0$

2. OR operator

To implement OR. let $w_i = 1$, $i = 1, 2, \dots, n$. we have $\begin{cases} w_0 + 0 \cdot n < 0 \\ w_0 + 1 > 0 \end{cases}$

$-1 < w_0 < 0$.

we have $w_0 = -0.5$. $w_i = 1$, $(i = 1, \dots n)$

$w = [-0.5 \quad 1 \quad 1 \quad \dots \quad 1]_{n+1}$

Only when every x_i equals to 0. can $y = Step(Xw) = Step(-0.5) = 0$.

even if only one $x_i \neq 0$. can $y = Step(Xw) = Step(-0.5 + 1) = 1$.

3. "Reverse"

Given that $n = 1$, $y = \neg x_1$. we have $\begin{cases} w_0 + 1 \cdot w_1 < 0 \\ w_0 + 0 \cdot w_1 > 0 \end{cases}$

$\begin{cases} w_1 < -w_0 \\ w_0 > 0 \end{cases}$ let $w_0 = 1$. $w_1 = -2$, $w = [1 \quad -2]$

when $x_1 = 1$. $y = Step(Xw) = Step(1 - 2) = 0$.

$x_1 = 0$, $y = Step(Xw) = Step(1) = 1$.

4. Majority vote

$\begin{cases} w_0 + \frac{n}{2} w_i < 0 \\ w_0 + (\frac{n}{2} + 1) w_i > 0 \end{cases} \Rightarrow \begin{cases} w_i > 0 \\ -\frac{n}{2} w_i < w_0 < -\frac{n}{2} w_i \end{cases}$ let $w_i = 1$. $w_0 = -\frac{n-1}{2}$

when $\frac{n}{2} + 1$ $x_i = 1$ $\Rightarrow$ $y = step(-\frac{n-1}{2} + \frac{n}{2} + 1) = step(\frac{1}{2}) = 1$

when $\frac{n}{2}$ $x_i = 1$ $\Rightarrow$ $y = step(0) = 0$.

Problem 4

In [19]:
```
import random
import numpy as np

# for easier reading np
np.set_printoptions(precision=3,suppress=True)
```

In [20]:
```
# Data is just a grid of points in 2D
X1, X2 = np.meshgrid(
    np.linspace(-1, 1, 20),
    np.linspace(-1, 1, 20)
)
X = np.stack((X1.flatten(), X2.flatten()), axis=1)
print('X', X.shape)
```

```
X (400, 2)
```

In [21]:
```
# Part a) Specify weights and biases.
w1 = np.array([[5, 2, -5],[-5, 5, -1]])
b1 = np.array([[1, -3, 4]])
w2 = np.array([[1],[1],[1]])
b2 = np.array([[-2.5]])
print('w1', w1.shape) # w1 should be a 2x3 matrix
print('b1', b1.shape) # b1 should be a 1x3 matrix or a 1d vector with 3 elements
print('w2', w2.shape) # w2 should be a 3x1 matrix
print('b2', b2.shape) # b2 should be a 1x1 matrix or a 1d vector with 1 element (basically a scalar)
```

```
w1 (2, 3)
b1 (1, 3)
w2 (3, 1)
b2 (1, 1)
```

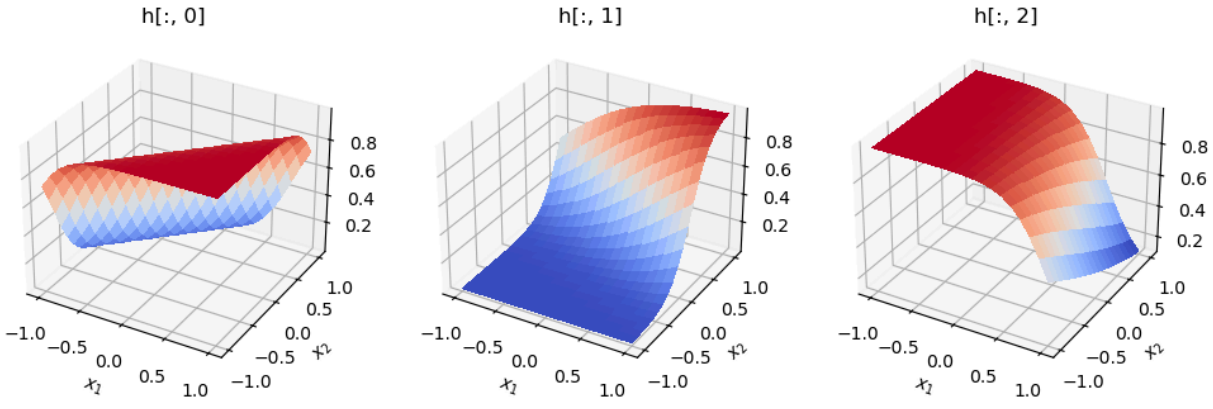
In [22]:
```
# Convert to your favorite library
import torch
X = torch.from_numpy(X).float()
w1 = torch.from_numpy(w1).float()
b1 = torch.from_numpy(b1).float()
w2 = torch.from_numpy(w2).float()
b2 = torch.from_numpy(b2).float()
```

In [23]:
```
# Part b) Compute the output of the network for inputs given by X.

# Hints: pytorch. Refer to:
# torch.sigmoid
# '@' implements matrix multiplication

# Compute outputs for hidden layer
h = torch.sigmoid(X @ w1 + b1)
print('h', h.shape)  # h should be a 400x3 matrix

# Compute outputs for output layer
z = torch.sigmoid(h @ w2 + b2)
print('z', z.shape)  # z should be a 400x1 matrix
```

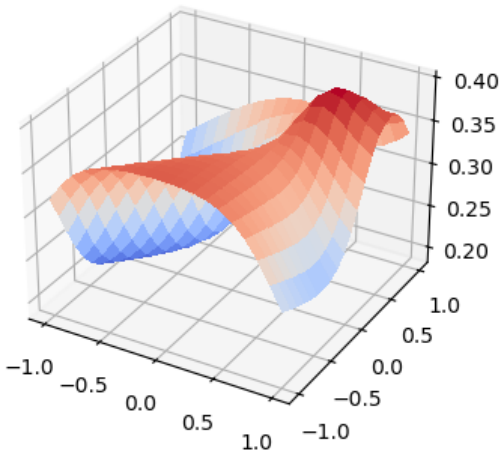
```
h torch.Size([400, 3])
z torch.Size([400, 1])
```

In [24]:
```
# Part c) Nothing to code.
from matplotlib import cm, pyplot as plt
fig, ax = plt.subplots(1, 3, figsize=(12, 4), subplot_kw={"projection": "3d"})
ax[0].plot_surface(X1, X2, h[:, 0].view(20, 20),
                   cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax[1].plot_surface(X1, X2, h[:, 1].view(20, 20),
                   cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax[2].plot_surface(X1, X2, h[:, 2].view(20, 20),
                   cmap=cm.coolwarm, linewidth=0, antialiased=False)
for i in range(3):
    ax[i].set_title(f'h[:, {i}]')
    ax[i].set_xlabel('$x_1$')
    ax[i].set_ylabel('$x_2$')

fig, ax = plt.subplots(figsize=(4, 4), subplot_kw={"projection": "3d"})
ax.plot_surface(X1, X2, z[:, 0].view(20, 20),
                   cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax.set_title('z=h[:,0]+h[:,1]+h[:,2]-2.5')
```

Out[24]:
```
Text(0.5, 0.92, 'z=h[:,0]+h[:,1]+h[:,2]-2.5')
```



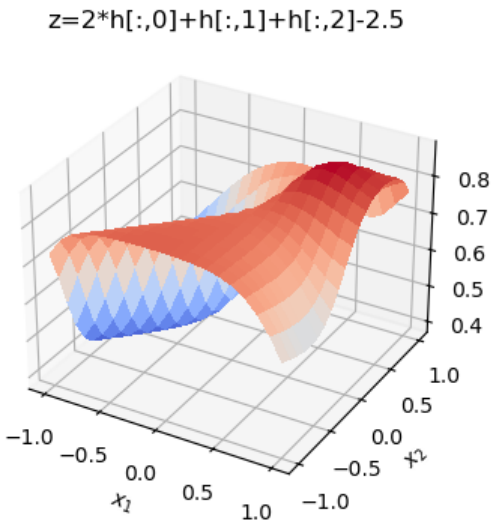
z=h[:,0]+h[:,1]+h[:,2]-2.5



Question: How do the directions of increase of the hidden neurons relate to the weight matrix?

Answer: The directions of increase of the hidden neurons are the directions of increase in the weight vector.

In [31]:
```
# Part d) How to increase output of the model at x=(1,-1) by changing the weights? Which weights to change, and h
w1 = torch.from_numpy(np.array([[7, 3, -5],[-7, 5, -1]])).float()
b1 = torch.from_numpy(np.array([[1, -3, 4]])).float()
w2 = torch.from_numpy(np.array([[2],[2],[2]])).float()
b2 = torch.from_numpy(np.array([[-2.5]])).float()

# Recompute outputs (reuse code from part b)
u1 = X @ w1 + b1
h = torch.sigmoid(u1)
u2 = h @ w2 + b2
z = torch.sigmoid(u2)

# Plot again to vizualize the results. Did the output at x=(1,-1) increased?
fig, ax = plt.subplots(figsize=(4, 4), subplot_kw={"projection": "3d"})
ax.plot_surface(X1, X2, z[:, 0].view(20, 20),
                cmap=cm.coolwarm, linewidth=0, antialiased=False)
ax.set_title('z=2*h[:,0]+h[:,1]+h[:,2]-2.5');
ax.set_xlabel('$x_1$');
ax.set_ylabel('$x_2$');
```



In [26]:
```
w1 = np.array([[5, 2, -5],[-5, 5, -1]])
b1 = np.array([[1, -3, 4]])
w2 = np.array([[1],[1],[1]])
b2 = np.array([[-2.5]])
```