Assignment 2

Q1: McCulloch-Pitts Neuron

In this question, I introduced the McCulloch-Pitts neuron, a simple mathematical model inspired by biological neurons. I drew the structure of a biological neuron and explained the correspondence:

- Dendrites correspond to input terminals.
- The cell body performs the summation of inputs.
- The axon transmits the output signal.
- Synapses represent the weights between neurons.

The McCulloch-Pitts neuron computes a weighted sum and applies a step function. I then showed how to build logic gates using this model:

- **AND gate**: weights = [1, 1], bias = -1.5
- **OR gate**: weights = [1, 1], bias = -0.5

Finally, I discussed why the **XOR gate cannot be implemented** with a single McCulloch-Pitts neuron — because XOR is not linearly separable.

🟩 Q2: MLP with Skip Connections (XOR Solver)

In this part, I analyzed a Multi-Layer Perceptron with a hidden neuron and a skip connection from input to output.

- I created a truth table showing that the network outputs correct XOR results.
- For $\mathbf{x_3}$, I derived the equation: $x_3 = \mathrm{step}(x_1 + x_2 - 1.5)$. I drew the decision boundary: a line $x_1 + x_2 = 1.5$.
- For $\mathbf{x_4}$, the output is $x_4 = \mathrm{step}(x_1 - x_2 + 0.5)$, so the decision line is $x_1 - x_2 = -0.5$.

These nonlinear boundaries together enable solving XOR.

🟩 Q3: Activation Functions

I introduced three popular activation functions:

- **Sigmoid**: outputs between 0 and 1
- **Tanh**: outputs between -1 and 1
- **ReLU**: outputs 0 for negative input, and x for positive input

I sketched all three on coordinate planes to show their behavior.

Q4: Shape of Parameters & Forward Pass

I described the matrix shapes of weights and biases in a standard feedforward neural network:

- **W1**: shape (d, h) — from input to hidden
- **b1**: shape (1, h)
- **W2**: shape (h, c) — from hidden to output
- **b2**: shape (1, c)

For input **x (1×d)**, I wrote:

```
f(x) = softmax(ReLU(xW1 + b1)W2 + b2)
```

And for batch input **X (n×d)**:

```
f(X) = softmax(ReLU(XW1 + b1)W2 + b2)
```

Programming Part

In Python, I implemented 3 neural networks:

1. **ANN1**: 2 input → 2 output (no hidden layer), with softmax
2. **ANN2**: 2 input → 8 hidden (no activation) → 2 output, with softmax
3. **ANN3**: 2 input → 8 hidden (ReLU) → 2 output, with softmax

I used `matplotlib.pyplot.contourf()` to visualize decision boundaries.

- ANN1 has linear boundaries due to no hidden layer.
- ANN2 shows more complex patterns than ANN1, but still limited because it lacks non-linearity.
- ANN3 performs best, showing nonlinear decision boundaries thanks to the ReLU activation in the hidden layer.

I concluded that **multi-layer networks with non-linear activation** are superior because they can approximate complex functions and classify data that are not linearly separable.

Assignment 4

Q1 – Key Idea of Backpropagation

> "The Backpropagation algorithm is a supervised learning technique used to train neural networks by minimizing the error between predicted and actual outputs.
> It consists of a forward pass to compute the output, and a backward pass to compute the gradients using the chain rule.
> These gradients are then used to update weights through gradient descent."

Q2

> "To answer Q2-2 efficiently, I wrote a Python script to simulate four steps of backpropagation training on a 1-2-1 MLP, using ReLU activation.
> This helped automate the calculations of intermediate values such as activations, errors, gradients, and weight updates."

> "Here's how the code works:"

◆ 1. Initialization

```
python复制编辑x = 0.7
d = 0.68
mu = 0.2
w1 = np.array([0.3, -0.3])
b1 = np.array([0.0, 0.0])
w2 = np.array([-0.1, 0.1])
b2 = 0.0
```

> "The input `x` is 0.7, and the target output `d` is 0.68.
> Weights and biases for the hidden layer (`w1`, `b1`) and output layer (`w2`, `b2`) are initialized manually.
> The learning rate `mu` is set to 0.2."

中文翻译:

> “输入 `x` 是 0.7，目标值 `d` 是 0.68。
> 我手动初始化了隐藏层的权重和偏置（`w1`, `b1`），以及输出层（`w2`, `b2`）。
> 学习率设为 0.2。”

◆ 2. Forward Pass and Backward Pass (Loop over 4 steps)

```
python复制编辑for t in range(steps):
    z1 = w1 * x + b1
    Z = relu(z1)
    z2 = np.dot(w2, Z) + b2
    y = z2
```

> "In each training step, I compute the hidden layer activation `Z` using ReLU, and the output `y` is computed as a linear combination of `Z` and weights `w2`."

🔹 3. Compute Errors and Gradients

```
python复制编辑delta2 = y - d
delta1 = relu_derivative(z1) * W2 * delta2

deltaW2 = mu * delta2 * z
deltaW1 = mu * delta1 * x
```

> "Then I compute the output error `delta2` and backpropagate it to get `delta1`.
> These are used to compute weight updates `deltaW1`, `deltaW2`, and corresponding bias updates."

中文翻译：

> “我计算输出误差 `delta2`，再通过链式法则计算 `delta1`。
> 然后用这些误差计算每一层的权重和偏置更新量。”

🔹 4. Update Weights and Log Results

```
python复制编辑W1 -= deltaW1
W2 -= deltaW2
b1 -= mu * delta1
b2 -= mu * delta2
```

> "After computing the gradients, I update the weights and biases.
> I also log all intermediate results such as weights, activations, deltas, and gradients in a table format."

中文翻译：

> “完成梯度计算后，我更新了每一层的权重和偏置。
> 我还把所有中间结果（如权重、激活值、误差、梯度等）记录到表格中。”

🔹 5. Result Table Output

> "The result is a clear, step-by-step table showing all variables for each training step, which matches the requirement of Q2-2.
> This approach ensures accurate, repeatable calculations and saved time compared to manual derivation."

中文翻译：

> “最终结果是一个清晰的表格，记录了每一轮训练的所有变量，完全符合Q2-2题目的要求。
> 相比手动推导，这种方法更高效、更准确，也易于复查。”

编程题

> "In the second part of the assignment, I implemented a neural network from scratch in Python using NumPy to solve the two-spiral classification task.
> I generated two classes of spiral data in 2D and used a fully connected network to separate them."

> "The network has:

- 2 inputs
- Two hidden layers (each with 16 units)
- 1 output (sigmoid for binary classification)"

> "I trained the model using the same BP principles I derived earlier."

中文翻译：

> “作业的第二部分是用Python和NumPy从零实现神经网络，以解决双螺旋分类问题。
> 我生成了两类二维螺旋数据，并使用一个前馈神经网络对其进行分类。”

> “网络结构包括：

- 两个输入
- 两层隐藏层（每层16个神经元）
- 一个输出节点（sigmoid用于二分类）”

> “我使用了和手推公式一致的反向传播方法进行训练。”

🔷 Slide 6: Code Details and Results

> "In the code:

- `train_nn()` runs forward and backward propagation for 2000 epochs
- `forward_pass()` is used to plot the decision boundary
- Loss was calculated using mean squared error"

> "I plotted:

- A training loss curve (which shows convergence)
- A decision boundary plot (showing the learned separation)"

中文翻译：

> “在代码中：

- `train_nn()` 函数执行前向和反向传播共2000轮
- `forward_pass()` 用于绘制决策边界
- 损失函数使用的是均方误差（MSE）”

> “我绘制了：

- 训练损失曲线（显示模型收敛过程）
- 决策边界图（显示模型对螺旋数据的分离效果）”

🔷 Slide 7: Reflection

> "It was challenging to implement the BP algorithm fully by hand.
> I had to carefully calculate each derivative and verify the gradient shapes.
> Normalizing the data and tuning the learning rate were crucial for successful training."

中文翻译：

> “手动实现完整的BP算法是有挑战性的。
> 我需要非常仔细地计算每一层的导数并确保梯度维度一致。
> 对数据进行归一化、调整学习率是训练成功的关键。”

Assignment 5

Q1 – Convolutional Neural Network Architecture

> "The question provided a CNN architecture with two convolutional layers, one pooling layer, and two fully connected layers.
> I calculated the number of weights and biases for each layer based on kernel sizes, channels, and units."

▪ Weights & Biases:

- Conv1: 4×4×3×8=3844×4×3×8 = 3844×4×3×8=384 weights + 8 biases
- Conv2: 8×8×8×16=81928×8×8×16 = 81928×8×8×16=8192 weights + 16 biases
- FC1: 1024×64=655361024×64 = 655361024×64=65536 weights + 64 biases
- FC2: 64×8=51264×8 = 51264×8=512 weights + 8 biases
  Total: 74,624 weights, 96 biases

▪ Output Shapes:

- Input: 84×84×3
- Conv1: 84×84×8
- Conv2: 26×26×16 (due to stride 3 and valid padding)
- Pooling: 8×8×16
- FC1: 64, FC2: 8, Softmax: 8

▪ Receptive Field:

- Final receptive field after pooling = **17×17**

Q2 – 1D Convolution Output Derivation

> "Given a 1D input signal and specified filters, I manually computed each output using valid padding, stride 1, and no activation."

▪ Conv1 Filters:

- Filter1 [0,1,0] → output: [2, 6, 7, 8, 9]
- Filter2 [0,0,1] → output: [6, 7, 8, 9, 1]
- Filter3 [1,0,0] → output: [1, 2, 6, 7, 8]
- Filter4 [1,0,1] → output: [7, 9, 14, 16, 9]

▪ Conv2:

- 1 filter, kernel size = 3×4, all weights = 1
- Result: [75, 99, 102]

Q3 – Unrolling Deep RNNs

> "For this question, I described how deep RNNs are unrolled through time.
> Each time step feeds into multiple RNN layers vertically.
> Horizontally, the hidden states propagate across time, and final outputs are taken from the last time step of the final layer."

编程

1. Importing Libraries

```
python复制编辑import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
```

> "I start by importing PyTorch and torchvision libraries.
> These provide modules for defining neural networks, activation functions, and data transformations."

中文翻译：

> “首先导入了 PyTorch 和 torchvision 库，它们提供了定义神经网络、激活函数、数据转换等必要的模块。”

---

2. Define the CNN Class

```
python复制编辑class MyCNN(nn.Module):
    def __init__(self):
        super(MyCNN, self).__init__()
        ...
```

> "I defined a class called `MyCNN`, which inherits from `nn.Module`.
> Inside the constructor `__init__`, I define all the layers of my convolutional neural network."

中文翻译：

> “我定义了一个类 `MyCNN`，继承自 PyTorch 的 `nn.Module`，
> 在构造函数 `__init__()` 中定义了CNN模型的所有网络层。”

---

3. Network Layers

```
python复制编辑self.conv1 = nn.Conv2d(1, 16, kernel_size=5, padding=2)
self.bn1 = nn.BatchNorm2d(16)
self.pool1 = nn.MaxPool2d(2)

self.conv2 = nn.Conv2d(16, 32, kernel_size=5, padding=2)
self.bn2 = nn.BatchNorm2d(32)
self.pool2 = nn.MaxPool2d(2)

self.fc1 = nn.Linear(32*7*7, 128)
self.dropout = nn.Dropout(0.5)
self.fc2 = nn.Linear(128, 10)
```

> "The CNN has:

- 2 convolutional layers (with batch normalization and max pooling)
- A fully connected layer with dropout
- A final output layer with 10 classes for MNIST"

> "I used kernel size = 5, and padding = 2 to maintain image size."

中文翻译：

> “模型包括：

- 两个卷积层（每个后接 BatchNorm 和 MaxPooling）
- 一个全连接层 + Dropout（防止过拟合）
- 最后一层为10个类别输出（用于MNIST）”

◆ 4. Forward Function

```
python复制编辑def forward(self, x):
    x = self.pool1(F.relu(self.bn1(self.conv1(x))))
    x = self.pool2(F.relu(self.bn2(self.conv2(x))))
    x = x.view(-1, 32 * 7 * 7)
    x = self.dropout(F.relu(self.fc1(x)))
    x = self.fc2(x)
    return x
```

> "The forward method defines how input data flows through the network:

1. Conv → BN → ReLU → Pool
2. Conv → BN → ReLU → Pool
3. Flatten
4. Fully connected layer + dropout
5. Output layer"

中文翻译：

> “`forward()` 函数定义了数据在网络中的流动方式：

1. 卷积 → BN → ReLU → 池化
2. 卷积 → BN → ReLU → 池化
3. 拉平为向量
4. 进入全连接层（带 Dropout）
5. 输出层给出分类结果”

◆ 5. Data Preprocessing

```
python复制编辑transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
trainset = torchvision.datasets.MNIST(...)
trainloader = torch.utils.data.DataLoader(...)
```

> "Here I downloaded the MNIST dataset and applied normalization using torchvision.
> Images are converted to tensors and pixel values are normalized to [-1, 1]."

中文翻译：

> “使用 `torchvision` 下载MNIST数据集，并进行标准化处理：
> 图像被转换为张量，并将像素值归一化到 [-1, 1] 区间。”

---

6. Training Setup

```
python复制编辑model = MyCNN()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

> "I used:

- `CrossEntropyLoss` as the loss function (suitable for multi-class classification)
- `Adam` optimizer with learning rate = 0.001"

中文翻译：

> “使用交叉熵损失函数（适用于多分类任务）和Adam优化器，学习率设置为0.001。”

---

7. Training Loop

```
python复制编辑for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(trainloader):
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

> "The model is trained for several epochs.
> In each iteration:

- Compute prediction
- Calculate loss
- Backpropagate and update weights"

中文翻译：

> “模型进行了多轮迭代训练，每轮包括：

- 前向传播计算输出
- 计算损失
- 反向传播并更新权重”

---

8. Evaluation and Accuracy

```
python复制编辑with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in testloader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()
```

> "After training, I evaluated the model accuracy on the test dataset.
> Predicted labels are compared with true labels to compute the total accuracy."

中文翻译：

> “训练完成后，我在测试集上评估模型准确率。
> 使用 `torch.max()` 获取最大概率对应的预测类别，并与真实标签进行比较。”