CS480/680 Spring 2026

Assignment 2

Designer: Abdenour Belarbi Instructor: Hongyang Zhang

Released: June 8 Due: **July 13, noon**

Instructions

- We do **not** accept hand-written submissions.
- For **coding** questions, implement in Python using a Jupyter Notebook. Ensure all cells run without error and that outputs are saved before submission.
- Submit **two files** to LEARN: a PDF write-up (written answers to *all* questions, including reported accuracy results for coding questions), and an IPYNB file (all coding questions with saved outputs).
- Name your files `Lastname_Firstname_assignment2.pdf` and `Lastname_Firstname_assignment2.ipynb`. Incorrect naming may result in a grade of 0.

Question 1 Theory (10 points)

1.1 Gradient Descent Convergence [3 pts]

Consider $f(x) = \frac{1}{2}(10x_1^2 + x_2^2)$, $x \in \mathbb{R}^2$.

(a) Compute the Hessian $\nabla^2 f(x)$. Is f convex? Is it strongly convex? Identify the smoothness constant L and the strong convexity constant m.

(b) What is the maximum step size t that guarantees convergence of gradient descent? Give the bound from the convex theorem ($t \leq 1/L$) and from the strongly convex theorem ($t \leq 2/(m+L)$).

(c) Assume $x^{(0)} = (1,1)^\top$ and target accuracy $\varepsilon = 0.01$. Using step size $t = 1/L$, find the minimum number of iterations k to guarantee $f(x^{(k)}) - f^* \leq \varepsilon$ under:

- the **convex** bound: $f(x^{(k)}) - f^* \leq \dfrac{\left\|x^{(0)} - x^*\right\|_2^2}{2tk}$
- the **strongly convex** bound: $f(x^{(k)}) - f^* \leq \gamma^k \dfrac{L}{2} \left\|x^{(0)} - x^*\right\|_2^2$, where $\gamma = 1 - m/L$.

1.2 CNN: Output Size and Parameter Counting [3 pts]

The following CNN is applied to FashionMNIST ($28 \times 28 \times 1$):

Layer	Type	Filters/Units	Kernel	Stride / Padding
1	Convolutional	16	5×5	stride 1, no padding
2	BatchNorm	–	–	–
3	Max Pooling	–	2×2	stride 2
4	Convolutional	32	3×3	stride 1, no padding
5	BatchNorm	–	–	–
6	Max Pooling	–	2×2	stride 2
7	Dropout ($p = 0.2$)	–	–	–
8	Fully Connected	10	–	–

Output size formula (height $\times$ width for one channel): $\left(1 + \frac{m-a}{s}\right) \times \left(1 + \frac{n-b}{s}\right)$ for input $m \times n$, kernel $a \times b$, stride s (no padding).

(a) Compute the output size (H×W×C) after *each* layer. Show your work.

(b) Count the learnable parameters (weights + biases) in each layer. For a conv layer: $a \cdot b \cdot c_{\text{in}} \cdot c_{\text{out}} + c_{\text{out}}$. For a BatchNorm layer with c channels: $2c$ (one γ and one β per channel). Pooling layers have *zero* parameters. Give the total.

(c) Compare the CNN's total parameters to a fully connected network with input 784, one hidden layer of 256 units, and output 10 (all with biases). What does this tell you about convolutional layers?

1.3 Transformer: Architecture and Scaling [4 pts]

Consider one Transformer encoder layer with $d = 256$, $h = 8$ heads, and feedforward dimension $d_{\text{ff}} = 1024$. Assume **no biases** in the attention projections and **biases included** in the feedforward and LayerNorm.

(a) Count the total learnable parameters in this encoder layer. Include: multi-head self-attention ($W_Q, W_K, W_V, W_O \in \mathbb{R}^{d \times d}$), feedforward sublayer ($W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, with biases), and the two LayerNorms ($\gamma, \beta \in \mathbb{R}^d$ each).

(b) Let $\mathbf{q}, \mathbf{k} \in \mathbb{R}^{d_k}$ have i.i.d. entries from $\mathcal{N}(0, 1)$. Show that $\text{Var}(\mathbf{q}^\top \mathbf{k}) = d_k$.

(c) As d_k grows, the dot products $\mathbf{q}^\top \mathbf{k}$ become large. Explain what this does to the softmax output and why it is a problem for training.

(d) Show that scaling by $1/\sqrt{d_k}$ restores the variance of the scores to 1.

Question 2 Backpropagation (8 points)

2.1 Softmax Gradient Warm-Up [1 pts]

A softmax output layer produces $\hat{\mathbf{p}} = \text{softmax}(\boldsymbol{\theta})$ and the loss is the cross-entropy $J = -\log \hat{p}_y$.

Recall from Lecture 4 that this can be written as $J = -\theta_y + \log \sum_j e^{\theta_j}$.

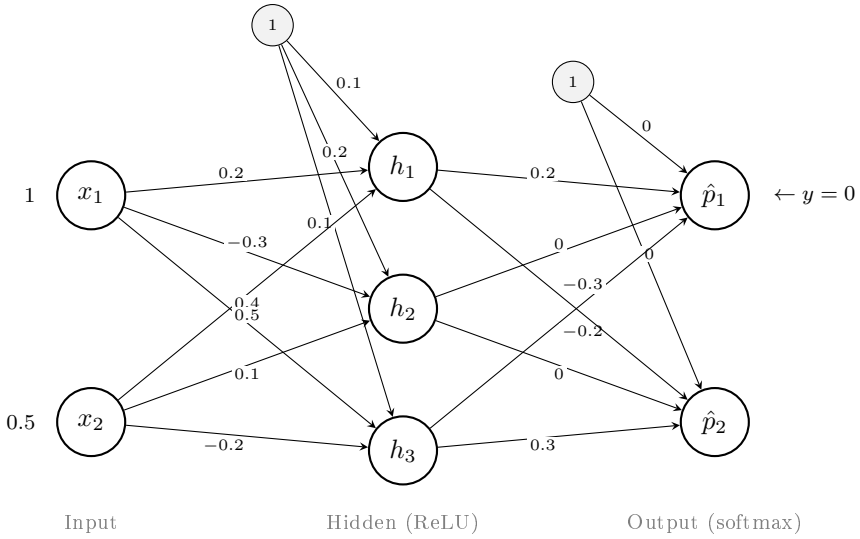
Given logits $\boldsymbol{\theta} = [0,\ 0.5,\ 0.25,\ 0.75]$ and true class $y = 2$:

(a) Compute $\hat{\mathbf{p}}$ and J (round to 4 decimal places).

(b) Show that $\dfrac{\partial J}{\partial \theta_i} = \hat{p}_i - \mathbf{1}[i = y]$, and evaluate this for the values above.

2.2 Full Backpropagation [7 pts]

Consider the 2-layer MLP below with all weights and bias terms labeled. The hidden layer uses ReLU; the output uses softmax with cross-entropy loss $J = -\log \hat{p}_y$. Input: $x_1 = 1$, $x_2 = 0.5$. True class: $y = 0$.



The same network written as explicit matrix operations (true class $y = 0$):

$$\mathbf{z} = \underbrace{\begin{pmatrix} 0.2 & 0.4 \\ -0.3 & 0.1 \\ 0.5 & -0.2 \end{pmatrix}}_{W} \underbrace{\begin{pmatrix} 1 \\ 0.5 \end{pmatrix}}_{\mathbf{x}} + \underbrace{\begin{pmatrix} 0.1 \\ 0.2 \\ 0.1 \end{pmatrix}}_{\mathbf{b}_1}, \quad \mathbf{h} = \mathrm{ReLU}(\mathbf{z}),$$

$$\boldsymbol{\theta} = \underbrace{\begin{pmatrix} 0.2 & 0 & -0.2 \\ -0.3 & 0 & 0.3 \end{pmatrix}}_{U} \mathbf{h} + \underbrace{\begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{\mathbf{b}_2}, \quad \hat{\mathbf{p}} = \mathrm{softmax}(\boldsymbol{\theta}), \quad J = -\log \hat{p}_0.$$

Hint: $\partial J / \partial X$ must have the same shape as X. Use this as a dimension check after each step.

(a) Compute the forward pass: $\mathbf{z}$, $\mathbf{h}$, $\boldsymbol{\theta}$, $\hat{\mathbf{p}}$, and J (exact form). Which hidden neuron is dead and why?

(b) Using Q2.1, state and compute $\partial J / \partial \boldsymbol{\theta}$.

(c) Derive and compute $\partial J / \partial U$ and $\partial J / \partial \mathbf{b}_2$. State dimensions.

(d) Derive and compute $\partial J / \partial \mathbf{h}$ and $\partial J / \partial \mathbf{z}$. Why is one entry of $\partial J / \partial \mathbf{z}$ zero?

(e) Derive and compute $\partial J / \partial W$ and $\partial J / \partial \mathbf{b}_1$. State dimensions.

(f) Perform one GD step with $t = 0.1$. Write the updated W^+ and U^+.

Question 3 Coding (7 points)

All models are trained on **FashionMNIST**. Use the data loader below for all sub-questions.

3.0 Data Loading (0 pts — do not modify)

```python
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

def get_FashionMNIST():
    torch.manual_seed(480)
    train_set = torchvision.datasets.FashionMNIST('./data', train=True,
                    transform=transforms.ToTensor(), download=True)
    test_set  = torchvision.datasets.FashionMNIST('./data', train=False,
                    transform=transforms.ToTensor())
    train_loader = DataLoader(train_set, batch_size=128, shuffle=True)
    test_loader  = DataLoader(test_set,  batch_size=128, shuffle=False)
    return train_loader, test_loader

train_loader, test_loader = get_FashionMNIST()
```

3.1 MLP [2 pts] (coding)

Implement and train an MLP on FashionMNIST.

Architecture: input 784 $\to$ Linear(256) $\to$ BatchNorm $\to$ ReLU $\to$ Dropout($p = 0.2$) $\to$ Linear(128) $\to$ BatchNorm $\to$ ReLU $\to$ Dropout($p = 0.2$) $\to$ Linear(10).

Training: `CrossEntropyLoss`, SGD (lr= 0.05, momentum= 0.9), 10 epochs.

Complete the skeleton and report the final training and test accuracy.

```python
import torch.nn as nn
import torch.optim as optim

##### COMPLETE THE FOLLOWING CODE #####
class MLP(nn.Module):
    def __init__(self, input_dim=784, hidden1=256, hidden2=128,
                 output_dim=10, dropout_p=0.2):
        super(MLP, self).__init__()
        # TODO: define layers
        pass

    def forward(self, x):
        # TODO: flatten input then apply layers
        pass

def compute_accuracy(model, loader, device):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            preds = model(images).argmax(dim=1)
            correct += (preds == labels).sum().item()
            total   += labels.size(0)
```

```
    return correct / total

def train(model, train_loader, test_loader,
          epochs=10, lr=0.05, device='cpu'):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=0.9)
    for epoch in range(epochs):
        model.train()
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            # TODO: forward pass, loss, backward, step
            pass
        tr = compute_accuracy(model, train_loader, device)
        te = compute_accuracy(model, test_loader,  device)
        print(f"Epoch {epoch+1:2d}  Train: {tr:.4f}  Test: {te:.4f}")

device = 'cuda' if torch.cuda.is_available() else 'cpu'
mlp = MLP()
train(mlp, train_loader, test_loader, device=device)
```

3.2 CNN [4 pts] (coding)

Implement and train the CNN from **Question 1.3** on FashionMNIST.

Architecture: Conv(16, 5×5) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ MaxPool(2×2) $\rightarrow$ Conv(32, 3×3) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ MaxPool(2×2) $\rightarrow$ Flatten $\rightarrow$ Dropout($p = 0.2$) $\rightarrow$ Linear(10).

Training: same settings as Q3.1 (reuse `train`).

Complete the skeleton and report the final training and test accuracy.

```
##### COMPLETE THE FOLLOWING CODE #####
class CNN(nn.Module):
    def __init__(self, num_classes=10):
        super(CNN, self).__init__()
        # TODO: define layers using nn.Conv2d, nn.BatchNorm2d,
        #       nn.MaxPool2d, nn.ReLU, nn.Flatten, nn.Dropout, nn.Linear
        pass

    def forward(self, x):
        # TODO: apply layers in order
        pass

cnn = CNN()
train(cnn, train_loader, test_loader, device=device)
```

3.3 Comparison [1 pts] (coding)

Fill in the table below with the final test accuracy of each model.

Model	Training Accuracy	Test Accuracy
MLP		
CNN		

In 2–3 sentences: which model (MLP or CNN) performs better, and give one architectural reason that explains the difference.

Grade breakdown: Q1.1: 3 pts Q1.2: 3 pts Q1.3: 4 pts Q2.1: 1 pt Q2.2: 7 pts Q3.1: 2 pts Q3.2: 4 pts Q3.3: 1 pt **Total: 25 pts**