CS/DS 541: Deep Learning

Homework 3

Tom Arnold & Nate Hindman

Due: 5:59pm ET Monday September 29

This problem can be done in teams of up 2 students.

Problem 1: Deriving He's initialization [10 points + 5 bonus pts]

In Lecture 7, we learned about a widely used technique for setting the initial values of the weight parameters in a neural network: the **He initialization**. Specifically, it samples each weight value from $\mathcal{N}(0, 2/n_l)$—i.e., a 0-mean Gaussian distribution with variance $2/n_l$ where n_l is the number of columns of the weight matrix $W^{[l]}$ layer l (or, equivalently the dimension of the inputs fed to layer l).

Here you are asked to derive some of the steps we skipped in class. We will start from:

$$Var[z^{[l]}] = Var[w^{[l]}x^{[l]}] = Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}]$$

$$Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}] = n_l Var[w_l x_l],$$

1. (2 points) Explain what are the assumptions that allow us to write:

$$Var[\sum_{j=1}^{n_l} w_j^{[l]}x_j^{[l]}] = n_l Var[w_l x_l],$$

where w_l represents any of the $w_j^{[l]}$ weights and x_l represents any of the $x_j^{[l]}$ inputs.

Answer

We are given that He initialization samples weights from $\mathcal{N}(0, 2/n_l)$ which means that each weight has expectation and variance:

$$E[w_j^{[l]}] = 0 \text{ and } \text{Var}[w_j^{[l]}] = 2/n_l \quad \text{(definition of } \mathcal{N}(\mu, \sigma^2))$$

$$\text{Var}\left[\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right] = E\left[\left(\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right)^2\right] - \left(E\left[\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right]\right)^2$$

$$\text{(Definition of Variance: } \text{Var}[Y] = E[Y^2] - (E[Y])^2)$$

Assumption 1: $w_j^{[l]}$ independent of all $x_k^{[l]}$

To move forward, we need to assume that the weights $w_j^{[l]}$ are independent of the inputs $x_j^{[l]}$. This assumption allows us to factor terms of the form $E[w_j^{[l]} x_j^{[l]}] = E[w_j^{[l]}] \cdot E[x_j^{[l]}]$. Because we know from initialization that $E[w_j^{[l]}] = 0$, the full product is 0 and drops out of the equation.

We are justified in this assumption because the random initialization of weights at layer l is done before the network ever sees data, so the particular input values flowing through the network cannot influence the weights. Without this assumption, we would have to deal with the full joint distribution of $(w_j^{[l]}, x_j^{[l]})$.

$$E\left[\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right] = \sum_{j=1}^{n_l} E[w_j^{[l]} x_j^{[l]}] \quad \text{(Linearity of Expectation)}$$

$$= \sum_{j=1}^{n_l} E[w_j^{[l]}] E[x_j^{[l]}] \quad \text{(Independence Theorem: } X \perp Y \Rightarrow E[XY] = E[X]E[Y])$$

$$= \sum_{j=1}^{n_l} 0 \cdot E[x_j^{[l]}] = 0$$

$$\text{Var}\left[\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right] = E\left[\left(\sum_{j=1}^{n_l} w_j^{[l]} x_j^{[l]}\right)^2\right] - 0^2$$

$$= E\left[\sum_{j=1}^{n_l} \sum_{k=1}^{n_l} w_j^{[l]} x_j^{[l]} w_k^{[l]} x_k^{[l]}\right] \quad \text{(Expanding } (a_1 + \cdots + a_n)^2 = \sum_j \sum_k a_j a_k)$$

$$= \sum_{j=1}^{n_l} \sum_{k=1}^{n_l} E[w_j^{[l]} x_j^{[l]} w_k^{[l]} x_k^{[l]}] \quad \text{(Linearity of Expectation)}$$

Assumption 2: $w_j^{[l]}$ independent of $w_k^{[l]}$ for $j \neq k$

When we expand the squared sum $(\sum_j w_j x_j)^2$, cross-terms of the form $E[w_j x_j w_k x_k]$ appear for $j \neq k$.

$$\text{Var}\left[\sum_{j=1}^{n_l} w_j x_j\right] = E\left[\begin{bmatrix} w_1 x_1 & \cdots & w_{n_l} x_{n_l} \end{bmatrix} \begin{bmatrix} w_1 x_1 \\ \vdots \\ w_{n_l} x_{n_l} \end{bmatrix}\right] = \begin{bmatrix} \text{Var}[w_1 x_1] & \cdots & E[w_j x_j w_k x_k] & \cdots \\ \vdots & & \ddots & \vdots \\ E[w_k x_k w_j x_j] & \cdots & \text{Var}[w_{n_l} x_{n_l}] & \end{bmatrix}$$

We can't move forward if these terms exist. Therefore we assume that the weights themselves are independent, meaning knowledge of $w_1^{[l]}$ tells us nothing about $w_2^{[l]}$, and so on. This independence causes these cross-terms to be 0, leaving only the diagonal contributions.

Thus, for $j \neq k$:

$$E[w_j^{[l]} x_j^{[l]} w_k^{[l]} x_k^{[l]}] = E[w_j^{[l]}] E[x_j^{[l]}] E[w_k^{[l]}] E[x_k^{[l]}] = 0 \cdot E[x_j^{[l]}] \cdot 0 \cdot E[x_k^{[l]}] = 0$$

(Independence of all four random variables)

$$= \sum_{j=1}^{n_l} E[(w_j^{[l]})^2 (x_j^{[l]})^2] \quad \text{(only diagonal terms } j = k \text{ survive)}$$

$$= \sum_{j=1}^{n_l} E[(w_j^{[l]} x_j^{[l]})^2]$$

$$= \sum_{j=1}^{n_l} \left(\text{Var}[w_j^{[l]} x_j^{[l]}] + (E[w_j^{[l]} x_j^{[l]}])^2\right) \quad \text{(Variance-Mean Relation: } E[Y^2] = \text{Var}[Y] + (E[Y])^2)$$

$$= \sum_{j=1}^{n_l} \text{Var}[w_j^{[l]} x_j^{[l]}] \quad \text{(since } E[w_j^{[l]} x_j^{[l]}] = E[w_j^{[l]}] E[x_j^{[l]}] = 0)$$

Assumption 3: All $w_j^{[l]}$ identically distributed, all $x_j^{[l]}$ identically distributed

To simplify further we must assume that all these weight–input pairs are identically distributed. This assumption is justified because under He initialization all weights are drawn from the same distribution, and the architecture of the network ensures that the inputs $x_j^{[l]}$ at a given layer have the same statistical properties. Without assuming identical distributions, each term in the summation could be different, forcing us to track them one by one rather than collapsing them as we do below.

$$= n_l \cdot \text{Var}[w^{[l]}x^{[l]}] \quad \text{(Identical Distribution: all terms equal)}$$

---

2. (4 points) Show that
$$Var[w_l x_l] = Var[w_l]\mathbb{E}[(x_l)^2]$$
using the following equality regarding the variance of the product of two mutually independent random variables A and B:
$$Var[AB] = Var[A]Var[B] + Var[A]\mathbb{E}[B]^2 + Var[B]\mathbb{E}[A]^2 \quad (0.0.1)$$
You will need to: (i) review and use the assumptions made about the distribution of w_l and (ii) use the relationship between variance and second moment for a random variable B:
$$Var[B] = \mathbb{E}[B^2] - \mathbb{E}[B]^2. \quad (0.0.2)$$

Answer

$$\text{Var}[w^{[l]}x^{[l]}] = E[(w^{[l]}x^{[l]})^2] - (E[w^{[l]}x^{[l]}])^2 \quad \text{(Definition of Variance)}$$

$$E[w^{[l]}x^{[l]}] = E[w^{[l]}] \cdot E[x^{[l]}] \quad \text{(Independence Theorem)}$$

$$= 0 \cdot E[x^{[l]}] = 0 \quad \text{(using } E[w^{[l]}] = 0 \text{ from He initialization)}$$

$$\text{Var}[w^{[l]}x^{[l]}] = E[(w^{[l]}x^{[l]})^2] - 0^2 = E[(w^{[l]})^2(x^{[l]})^2]$$

$$= E[(w^{[l]})^2] \cdot E[(x^{[l]})^2] \quad \text{(Independence Theorem for products)}$$

$$E[(w^{[l]})^2] = \text{Var}[w^{[l]}] + (E[w^{[l]}])^2 \quad \text{(Variance-Mean Relation)}$$

$$= \text{Var}[w^{[l]}] + 0^2 = \text{Var}[w^{[l]}]$$

$$\therefore \text{Var}[w^{[l]}x^{[l]}] = \text{Var}[w^{[l]}] \cdot E[(x^{[l]})^2]$$

---

3. Last, you need to relate the variance of the input x_l with the variance of the pre-activation value z_{l-1} when the activation function is **ReLU**, i.e. when $x_l = ReLU(z_{l-1}) = ReLU(w_{l-1}x_{l-1} + b_{l-1})$. Specifically, you need to prove the relationship:

$$Var[z_{l-1}] = 2 \times \mathbb{E}[x_l^2].$$

To do so, you will need to use these assumptions:

- w_{l-1} has a **symmetric distribution around 0**, i.e., the density $f_{w_{l-1}}(w) = f_{w_{l-1}}(-w)$.
- $b_{l-1} = 0$

As a guide, try to answer these questions in order:

(a) (2 points) What is the expected value of z_{l-1}? Does it depend on the value of x_{l-1}?

(b) (1 point) Think of x_{l-1} as a constant. What is the density $f_{z_{l-1}}(z)$!

(c) (1 point) Is the distribution of z_{l-1} symmetric around 0?

(d) (BONUS: 3 points) The variance of a continuous random variable X is given by $\int_{-\infty}^{\infty}(x - \mathbb{E}[X])^2 f_X(x)dx$. Write an expression for $Var[z_{l-1}]$. Remember what you found in item (a) about $\mathbb{E}[z_{l-1}]$ and try to use the symmetry of the distribution to write $Var[z_{l-1}]$ only in terms of the positive values of z_{l-1}.

(e) (BONUS: 2 points) Last, starting from the definition of the second moment $\mathbb{E}[x_l^2] = \int_{-\infty}^{\infty} x_l^2 f_{x_l}(x)d_x$, find ways to replace x_l by z_{l-1} conditioning on whether z_{l-1} is negative or positive. You should be able to find the expression you derived in item (d) for $Var[z_{l-1}]$.

Answer

(a) Expected value of z_{l-1}:

Given: $b_{l-1} = 0$

$$z_{l-1} = \sum_{k=1}^{n_{l-1}} w_k^{[l-1]} x_k^{[l-1]}$$

$$E[z_{l-1}] = E\left[\sum_{k=1}^{n_{l-1}} w_k^{[l-1]} x_k^{[l-1]}\right] = \sum_{k=1}^{n_{l-1}} E[w_k^{[l-1]} x_k^{[l-1]}] \quad \text{(Linearity of Expectation)}$$

$$= \sum_{k=1}^{n_{l-1}} E[w_k^{[l-1]}] \cdot E[x_k^{[l-1]}] \quad \text{(Independence Theorem)}$$

$$= \sum_{k=1}^{n_{l-1}} 0 \cdot E[x_k^{[l-1]}] = 0 \quad \text{(He initialization: } E[w_k^{[l-1]}] = 0\text{)}$$

(b) Density of z_{l-1} given x_{l-1}:

$$z_{l-1} = w_{l-1} \cdot x_{l-1} \quad \text{(treating } x_{l-1} \text{ as constant scalar)}$$

$$f_{z_{l-1}|x_{l-1}}(z) = \frac{1}{|x_{l-1}|} f_{w_{l-1}}\left(\frac{z}{x_{l-1}}\right)$$

(Change of Variables Formula: if $Y = aX$, then $f_Y(y) = \frac{1}{|a|} f_X(y/a)$)

(c) Symmetry:

Given: $w_{l-1} \sim \mathcal{N}(0, \sigma^2)$

$$\Rightarrow f_{w_{l-1}}(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-w^2/(2\sigma^2)} = f_{w_{l-1}}(-w)$$

(Gaussian PDF is symmetric about its mean)

$$f_{z_{l-1}}(z) = f_{z_{l-1}}(-z) \quad \text{(Linear transformation preserves symmetry)}$$

(d) Variance calculation:

$$\text{Var}[z_{l-1}] = E[z_{l-1}^2] - (E[z_{l-1}])^2 = E[z_{l-1}^2] - 0 \quad \text{(Definition of Variance)}$$

$$E[z_{l-1}^2] = \int_{-\infty}^{\infty} z^2 f_{z_{l-1}}(z)dz \quad \text{(Definition of Expectation for continuous RV)}$$

$$= \int_{-\infty}^{0} z^2 f_{z_{l-1}}(z)dz + \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz \quad \text{(Additivity of Integrals)}$$

$$= \int_{0}^{\infty} u^2 f_{z_{l-1}}(-u)du + \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz$$

(U-Substitution Rule for Integrals: Let $u = -z$ in first integral, then $du = -dz$)

$$= \int_{0}^{\infty} u^2 f_{z_{l-1}}(u)du + \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz$$

(Using symmetry: $f_{z_{l-1}}(-u) = f_{z_{l-1}}(u)$)

$$= 2\int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz \quad \text{(Both integrals are identical)}$$

(e) Connecting to $E[x_l^2]$:

$$x_l = \text{ReLU}(z_{l-1}) = \begin{cases} z_{l-1} & \text{if } z_{l-1} > 0 \\ 0 & \text{if } z_{l-1} \leq 0 \end{cases} \quad \text{(Definition of ReLU)}$$

$$E[x_l^2] = \int_{-\infty}^{\infty} [\text{ReLU}(z)]^2 f_{z_{l-1}}(z)dz \quad \text{(Definition of Expectation)}$$

$$= \int_{-\infty}^{0} 0^2 \cdot f_{z_{l-1}}(z)dz + \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz$$

($\text{ReLU}(z) = 0$ for $z \leq 0$, $\text{ReLU}(z) = z$ for $z > 0$)

$$= 0 + \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz = \int_{0}^{\infty} z^2 f_{z_{l-1}}(z)dz$$

From part (d): $\text{Var}[z_{l-1}] = 2\int_0^\infty z^2 f_{z_{l-1}}(z)dz$

$$\therefore \text{Var}[z_{l-1}] = 2 \cdot E[x_l^2]$$

Problem 2: Training NNs with pytorch [15 points + 2 bonus pts]

In this problem you will use pytorch to train a multi-layer neural network to classify images of fashion items (10 different classes) from the **Fashion MNIST** dataset. Similarly to Homework 2, the input to the network will be a 28×28 pixel image; the output will be a real number.

Specifically, the starting network you will create should implement a function $f : \mathbb{R}^{784} \to \mathbb{R}^{10}$, where:

$$\begin{aligned} z^{(1)} &= W^{(1)}x + b^{(1)} \\ h^{(1)} &= ReLU(z^{(1)}) \\ z^{(2)} &= W^{(2)}h^{(1)} + b^{(2)} \\ &\vdots \\ z^{(l)} &= W^{(l)}h^{(l-1)} + b^{(l)} \\ \hat{y} &= softmax(z^{(l)}) \end{aligned}$$

The network specified above is shown in the figure below:

As usual, the (unregularized) cross-entropy cost function should be:

$$f_{CE}(W^{(1)}, b^{(1)}, ..., W^{(l)}, b^{(l)}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{10} y_k^{(i)} log\hat{y}_k^{(i)}$$

where n is the number of examples.

The Fashion MNIST dataset can be obtained from the following web links: https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_images.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_train_labels.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_images.npy https://s3.amazonaws.com/jrwprojects/fashion_mnist_test_labels.npy

1. (6 points) Implement the same network using PyTorch or Tensorflow.
2. (5 points) Implement a method called **hyperparam_tuning**. Optimize the hyperparameters by training on the training set and selecting the parameter settings that optimize performance on the validation set. You should systematically (i.e., in code) try at least **10** (in total, not for each hyperparameter) different hyperparameter settings; **Hyperparameter tuning:** In this problem, there are several different hyperparameters that will impact the network's performance:

- (Required) **Number of hidden layers** (suggestions: $\{3, 4, 5\}$)
- **Number of units in each hidden layer** (suggestions: $\{30, 40, 50\}$)
- **Learning rate** (suggestions: $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$)
- **Minibatch size** (suggestions: $\{16, 32, 64, 128, 256\}$)
- **Number of epochs**
- L_2 **Regularization strength** applied to the weight matrices (but not bias terms) In order not to "cheat" and thus overestimate the performance of the network it is crucial to optimize the hyperparameters **only on the validation set**; do **not** use the test set. (The training set would be ok, but typically leads to worse performance.) Hence, just like in Homework 2, you should fork off part of the training set into a validation portion.

3. (4 points) After you have optimized your hyperparameters, then re-train your network again and show a screenshot displaying the training loss evolving over the last **20 iterations** of SGD. (You can pick the last 20 mini-batches or last 20 epochs; it's up to you). In addition, make sure your screenshot also displays the **test accuracy** of the final trained classifier. The accuracy (percentage correctly classified test images) should be **at least 87%**.

4. (2 bonus points) Now you have to experiment with a variant of the previous network that includes either **BatchNorm, Dropout layers** or **different activation functions** (e.g., LeakyReLU, ELU or PReLU). Find an architecture that outperforms the one you found in item 3 and report the test accuracy.

Problem 3: Visualizing the loss landscape and optimization trajectories (15 points)

Visualize the SGD trajectory of your network when trained on Fashion MNIST:

1. Plot in 3-D the cross-entropy loss f_{CE} as a function of the neural network parameters (weights and bias terms). Rather than showing the loss as a function of any particular parameters (e.g., the third component of $b^{(2)}$), use the **x and y-axes** to span the two directions along which your parameters vary the most during SGD training i.e., you will need to use **principal component analysis (PCA)**. (In this assignment, you are free to use `sklearn.decomposition.PCA`.) The **z-axis** should represent the (unregularized) f_{CE} on training data. For each point (x, y) on a grid, compute f_{CE} and then interpolate between the points. (The interpolation and rendering are handled for you completely by the `plot_surface` function; see the starter code).

2. Superimpose a scatter plot of the different points (in the neural network's parameter space) that were reached during SGD (just sample a few times per epoch). To accelerate the rendering, train the network and plot the surface using just a small subset of the training data (e.g., 2500 examples) to estimate f_{CE}. See the starter code, in particular the `plotPath` function, for an example of 3-D plotting. Submit your graph in the PDF file and the code in the Python file.

   Here is what I get when I superimpose two scatter plots corresponding to two different initializations and SGD runs (note that you only need to include a scatter plot for one run); as you can see, the two SGD trajectories descended into distinct local minima (though with similar cost):

Problem 4: Autoencoder (20 points)

This assignment explores one powerful concept in deep learning: **unsupervised representation learning with autoencoders**. You will build a neural network that learns to compress and then reconstruct images, forcing it to learn a meaningful representation of the data in the process.

Task 1: Data Preparation (2 point)

1. Load the **Fashion MNIST** dataset using `torchvision.datasets.FashionMNIST`.
2. Use `torchvision.transforms.ToTensor` to convert the images to PyTorch tensors and normalize the pixel values to be in the range $[0.0, 1.0]$.
3. Create a `DataLoader` for both the training and test sets. For this unsupervised task, we only need the image data, not the labels.

Answer

```
# Task 1: Load Fashion-MNIST, convert to tensors in [0,1], make DataLoaders.

def prepare_data(batch_size: int = 128):
    transform = transforms.ToTensor()  # 28x28 grayscale -> float tensor in
 ↪  [0,1]
```

```
    train_set = torchvision.datasets.FashionMNIST(
        root="./", train=True, download=True, transform=transform
    )
    test_set = torchvision.datasets.FashionMNIST(
        root="./", train=False, download=True, transform=transform
    )

    train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True,
↪   num_workers=2)
    test_loader  = DataLoader(test_set,  batch_size=batch_size,
↪   shuffle=False, num_workers=2)

    # quick checks
    print("n/Data ready")
    print(f"- train samples: {len(train_set)}")
    print(f"- test samples : {len(test_set)}")
    print(f"- image shape  : {train_set[0][0].shape}")  # (1, 28, 28)
    print(f"- train batches: {len(train_loader)}")
    return train_loader, test_loader

# run task 1
train_loader, test_loader = prepare_data(batch_size=128)
```

```
n/Data ready
- train samples: 60000
- test samples : 10000
- image shape  : torch.Size([1, 28, 28])
- train batches: 469
```

Task 2: Autoencoder Architecture (6 points)

Construct a **dense autoencoder model** by creating a class that inherits from `torch.nn.Module`. The model should consist of an **encoder** and a **decoder** with the following symmetric architecture:

- **An Encoder** that takes a flattened 784-dimensional input and maps it to a compressed representation.
  - Input layer (implicitly defined by the input shape).

  - Dense layer (`nn.Linear`) with **128 units** followed by a **ReLU** activation (`nn.ReLU`).
  - Dense layer with **64 units** followed by a **ReLU** activation. This layer's output is the **latent representation**, also known as the "**bottleneck**".

- **A Decoder** that takes the 64-dimensional latent representation and reconstructs the 784-dimensional image.
  - Dense layer with **128 units** followed by a **ReLU** activation.
  - Dense layer with **784 units** followed by a **Sigmoid** activation (`nn.Sigmoid`). The sigmoid activation ensures the output values are in the range $[0.0, 1.0]$, matching the normalized input data.

Answer

```
# Task 2: Dense autoencoder. Encoder: 784->128->64. Decoder: 64->128->784
 ↪  with Sigmoid.

class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 128), nn.ReLU(),
            nn.Linear(128, 64),  nn.ReLU(),
        )
        self.decoder = nn.Sequential(
            nn.Linear(64, 128),  nn.ReLU(),
            nn.Linear(128, 784), nn.Sigmoid(),  # keep outputs in [0,1]
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.decoder(self.encoder(x))

    def encode(self, x: torch.Tensor) -> torch.Tensor:
        return self.encoder(x)

# build model and show size
model = Autoencoder()
params = sum(p.numel() for p in model.parameters())
print("Model ready")
```

```
print(f"- parameters: {params:,}")
```

```
Model ready
- parameters: 218,192
```

Task 3: Model Training (6 points)

1. Instantiate your model, a loss function (`nn.MSELoss` is a good choice), and an optimizer (`torch.optim.Adam`).
2. Write a training loop that iterates for **20 epochs**. In each epoch, iterate through the training `DataLoader`.
3. For each batch of images, you must:
    - Flatten the 28×28 images into 784-dimensional vectors.
    - Perform a forward pass to get the reconstructed images.
    - Calculate the loss between the original and reconstructed images.
    - Zero the gradients, perform a backward pass, and update the weights.

Answer

```
# Task 3: Train for 20 epochs with MSE loss and Adam.

def train_autoencoder(model: nn.Module, loader: DataLoader, epochs: int = 20,
 ↪  lr: float = 1e-3):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)
    opt = optim.Adam(model.parameters(), lr=lr)
    loss_fn = nn.MSELoss()

    print(f"Training on {device} for {epochs} epochs")
    losses = []

    for epoch in range(1, epochs + 1):
```

```
        model.train()
        running = 0.0

        for i, (imgs, _) in enumerate(loader):
            imgs = imgs.to(device).view(imgs.size(0), -1)   # (B, 1, 28, 28)
 ↪  -> (B, 784)

            opt.zero_grad()
            out = model(imgs)
            loss = loss_fn(out, imgs)
            loss.backward()
            opt.step()

            running += loss.item()
            if i % 100 == 0:
                print(f"  epoch {epoch:2d} | batch {i:4d}/{len(loader)} |
                 ↪  loss {loss.item():.4f}")

        avg = running / len(loader)
        losses.append(avg)
        print(f"  epoch {epoch:2d} complete | avg loss {avg:.4f}")

    return model, losses

# run task 3
model, losses = train_autoencoder(model, train_loader, epochs=20, lr=1e-3)
```

```
Training on cuda for 20 epochs
  epoch  1 | batch    0/469 | loss 0.1714
  epoch  1 | batch  100/469 | loss 0.0410
  epoch  1 | batch  200/469 | loss 0.0289
  epoch  1 | batch  300/469 | loss 0.0250
  epoch  1 | batch  400/469 | loss 0.0228
  epoch  1 complete | avg loss 0.0374
  epoch  2 | batch    0/469 | loss 0.0212
  epoch  2 | batch  100/469 | loss 0.0196
  epoch  2 | batch  200/469 | loss 0.0210
  epoch  2 | batch  300/469 | loss 0.0218
  epoch  2 | batch  400/469 | loss 0.0195
  epoch  2 complete | avg loss 0.0198
  epoch  3 | batch    0/469 | loss 0.0173
```

```
epoch  3 | batch  100/469 | loss 0.0168
epoch  3 | batch  200/469 | loss 0.0161
epoch  3 | batch  300/469 | loss 0.0163
epoch  3 | batch  400/469 | loss 0.0155
epoch  3 complete | avg loss 0.0172
epoch  4 | batch    0/469 | loss 0.0167
epoch  4 | batch  100/469 | loss 0.0159
epoch  4 | batch  200/469 | loss 0.0151
epoch  4 | batch  300/469 | loss 0.0149
epoch  4 | batch  400/469 | loss 0.0153
epoch  4 complete | avg loss 0.0156
epoch  5 | batch    0/469 | loss 0.0158
epoch  5 | batch  100/469 | loss 0.0156
epoch  5 | batch  200/469 | loss 0.0147
epoch  5 | batch  300/469 | loss 0.0141
epoch  5 | batch  400/469 | loss 0.0151
epoch  5 complete | avg loss 0.0145
epoch  6 | batch    0/469 | loss 0.0133
epoch  6 | batch  100/469 | loss 0.0143
epoch  6 | batch  200/469 | loss 0.0126
epoch  6 | batch  300/469 | loss 0.0132
epoch  6 | batch  400/469 | loss 0.0135
epoch  6 complete | avg loss 0.0136
epoch  7 | batch    0/469 | loss 0.0130
epoch  7 | batch  100/469 | loss 0.0145
epoch  7 | batch  200/469 | loss 0.0117
epoch  7 | batch  300/469 | loss 0.0120
epoch  7 | batch  400/469 | loss 0.0115
epoch  7 complete | avg loss 0.0129
epoch  8 | batch    0/469 | loss 0.0125
epoch  8 | batch  100/469 | loss 0.0124
epoch  8 | batch  200/469 | loss 0.0116
epoch  8 | batch  300/469 | loss 0.0124
epoch  8 | batch  400/469 | loss 0.0126
epoch  8 complete | avg loss 0.0124
epoch  9 | batch    0/469 | loss 0.0135
epoch  9 | batch  100/469 | loss 0.0118
epoch  9 | batch  200/469 | loss 0.0108
epoch  9 | batch  300/469 | loss 0.0125
epoch  9 | batch  400/469 | loss 0.0126
epoch  9 complete | avg loss 0.0119
epoch 10 | batch    0/469 | loss 0.0112
epoch 10 | batch  100/469 | loss 0.0115
```

```
epoch 10 | batch  200/469 | loss 0.0111
epoch 10 | batch  300/469 | loss 0.0117
epoch 10 | batch  400/469 | loss 0.0115
epoch 10 complete | avg loss 0.0116
epoch 11 | batch    0/469 | loss 0.0104
epoch 11 | batch  100/469 | loss 0.0115
epoch 11 | batch  200/469 | loss 0.0109
epoch 11 | batch  300/469 | loss 0.0115
epoch 11 | batch  400/469 | loss 0.0108
epoch 11 complete | avg loss 0.0112
epoch 12 | batch    0/469 | loss 0.0111
epoch 12 | batch  100/469 | loss 0.0114
epoch 12 | batch  200/469 | loss 0.0109
epoch 12 | batch  300/469 | loss 0.0108
epoch 12 | batch  400/469 | loss 0.0099
epoch 12 complete | avg loss 0.0109
epoch 13 | batch    0/469 | loss 0.0110
epoch 13 | batch  100/469 | loss 0.0109
epoch 13 | batch  200/469 | loss 0.0101
epoch 13 | batch  300/469 | loss 0.0124
epoch 13 | batch  400/469 | loss 0.0093
epoch 13 complete | avg loss 0.0107
epoch 14 | batch    0/469 | loss 0.0113
epoch 14 | batch  100/469 | loss 0.0112
epoch 14 | batch  200/469 | loss 0.0105
epoch 14 | batch  300/469 | loss 0.0104
epoch 14 | batch  400/469 | loss 0.0102
epoch 14 complete | avg loss 0.0105
epoch 15 | batch    0/469 | loss 0.0115
epoch 15 | batch  100/469 | loss 0.0096
epoch 15 | batch  200/469 | loss 0.0098
epoch 15 | batch  300/469 | loss 0.0101
epoch 15 | batch  400/469 | loss 0.0123
epoch 15 complete | avg loss 0.0103
epoch 16 | batch    0/469 | loss 0.0100
epoch 16 | batch  100/469 | loss 0.0114
epoch 16 | batch  200/469 | loss 0.0111
epoch 16 | batch  300/469 | loss 0.0105
epoch 16 | batch  400/469 | loss 0.0103
epoch 16 complete | avg loss 0.0101
epoch 17 | batch    0/469 | loss 0.0092
epoch 17 | batch  100/469 | loss 0.0106
epoch 17 | batch  200/469 | loss 0.0095
```

```
epoch 17 | batch  300/469 | loss 0.0099
epoch 17 | batch  400/469 | loss 0.0101
epoch 17 complete | avg loss 0.0100
epoch 18 | batch    0/469 | loss 0.0099
epoch 18 | batch  100/469 | loss 0.0101
epoch 18 | batch  200/469 | loss 0.0093
epoch 18 | batch  300/469 | loss 0.0105
epoch 18 | batch  400/469 | loss 0.0097
epoch 18 complete | avg loss 0.0098
epoch 19 | batch    0/469 | loss 0.0101
epoch 19 | batch  100/469 | loss 0.0104
epoch 19 | batch  200/469 | loss 0.0094
epoch 19 | batch  300/469 | loss 0.0089
epoch 19 | batch  400/469 | loss 0.0085
epoch 19 complete | avg loss 0.0097
epoch 20 | batch    0/469 | loss 0.0097
epoch 20 | batch  100/469 | loss 0.0088
epoch 20 | batch  200/469 | loss 0.0108
epoch 20 | batch  300/469 | loss 0.0094
epoch 20 | batch  400/469 | loss 0.0093
epoch 20 complete | avg loss 0.0095
```

Task 4: Visualizing Reconstructions (6 points)

1. Set your model to **evaluation mode**.
2. Get a batch of images from the test set and pass them through your trained autoencoder to get the reconstructions.
3. Create a plot showing **two different original images** from the test set and their corresponding reconstructions directly below them.
4. **Deliverable:** A single figure containing 4 images (2 original test images, 2 reconstructed images), clearly labeled.

Answer

```
# Task 4: Eval mode, pass a test batch, show 2 originals (top) and 2
 ↪  reconstructions (bottom).

def visualize_reconstructions(model: nn.Module, test_loader: DataLoader,
 ↪  num_images: int = 2):
    model.eval()
    device = next(model.parameters()).device

    images, _ = next(iter(test_loader))
    images = images.to(device)
    with torch.no_grad():
        recon = model(images.view(images.size(0), -1)).view(-1, 1, 28, 28)

    images = images.cpu()
    recon = recon.cpu()

    fig, axes = plt.subplots(2, 2, figsize=(8, 4))
    axes[0, 0].imshow(images[0].squeeze(), cmap="gray"); axes[0,
 ↪  0].set_title("Original 1"); axes[0, 0].axis("off")
    axes[1, 0].imshow(recon[0].squeeze(),   cmap="gray"); axes[1,
 ↪  0].set_title("Reconstructed 1"); axes[1, 0].axis("off")
    axes[0, 1].imshow(images[1].squeeze(), cmap="gray"); axes[0,
 ↪  1].set_title("Original 2"); axes[0, 1].axis("off")
    axes[1, 1].imshow(recon[1].squeeze(),   cmap="gray"); axes[1,
 ↪  1].set_title("Reconstructed 2"); axes[1, 1].axis("off")
    fig.suptitle("Autoencoder: Test Originals vs Reconstructions")
    plt.tight_layout()
    plt.show()

# run task 4
visualize_reconstructions(model, test_loader, num_images=2)
```

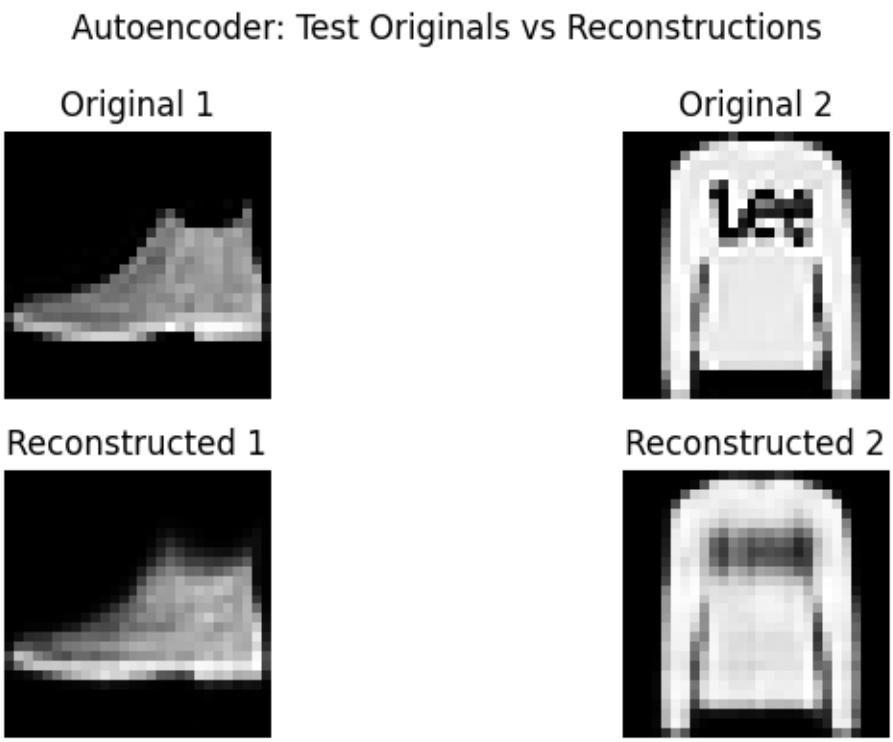

Figure 1: Autoencoder reconstructions on test set: Originals (top) vs Reconstructions (bottom).

Submission

Submit one PDF file that includes your notes for the theoretical problems (scanned or typed) and screenshots of your code for the programming problems. All material in the submitted PDF must be presented in a clear and readable format.

If you are working as part of a group, then indicate the members on canvas: https://canvas.wpi.edu/courses/7677
14853. Once you do that, be aware that any submission from a team member will overwrite an existing one.

Teamwork

You may complete this homework assignment either individually or in teams up to 2 people.