CS/DS 541: Deep Learning

Homework 2

Due: 5:59pm ET Thursday September 18

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

Problem 1: Derivation of softmax regression gradient [15 points]

As explained in class, the softmax regression generalizes the logistic regression to multi-class classification. Let

$$W = \left[w^{(1)} \ \dots \ w^{(c)} \right]$$

be an $m \times c$ matrix containing the weight vectors from the c different classes. The output of the softmax regression neural network is a vector with c dimensions such that:

$$\hat{y}_k = \frac{\exp(z_k)}{\sum_{k'=1}^{c} \exp(z_{k'})} \tag{0.0.1}$$

$$z_k = x^\top w^{(k)} + b_k$$

for each $k = 1, \dots, c$. Correspondingly, our cost function will sum over all c classes:

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

Important note: When deriving the gradient expression for each weight vector $w^{(l)}$, it is crucial to keep in mind that the weight vector for each class $l \in \{1, \dots, c\}$ affects the outputs of the network for every class, not just for class l. This is due to the normalization in Equation 0.0.1 – if changing the weight vector increases the value of $\hat{y}_l$, then it necessarily must decrease the values of the other $\hat{y}_{l' \neq l}$.

In this homework problem, please complete the following derivation that is outlined below:

Derivation: For each weight vector $w^{(l)}$, we can derive the gradient expression as:

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

$$= -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \left(\frac{\nabla_{w^{(l)}} \hat{y}_k^{(i)}}{\hat{y}_k^{(i)}} \right)$$

We handle the two cases $l = k$ and $l \neq k$ separately.

For $l = k$:

$$\nabla_{w^{(l)}} \hat{y}_k^{(i)} = \text{complete me...}$$

$$= x^{(i)} \hat{y}_l^{(i)} (1 - \hat{y}_l^{(i)})$$

For $l \neq k$:

$$\nabla_{w^{(l)}} \hat{y}_k^{(i)} = \text{complete me...}$$

$$= -x^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)}$$

To compute the total gradient of f_{CE} w.r.t. each $w^{(k)}$, we have to sum over all examples and over $l = 1, \dots, c$. (Hint: $\sum_k a_k = a_l + \sum_{k \neq l} a_k$. Also, $\sum_k y_k = 1$.)

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

$$= \text{complete me...}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} x^{(i)} \left(y_l^{(i)} - \hat{y}_l^{(i)} \right)$$

Finally, show that:

$$\nabla_b f_{CE}(W, b) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)$$

Answers:

Step 1 $\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)}$

$$\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} = \frac{\partial \hat{y}_k^{(i)}}{\partial z_l^{(i)}} \cdot \frac{\partial z_l^{(i)}}{\partial \mathbf{w}^{(l)}} \quad \text{(chain rule)}$$

$$\frac{\partial z_l^{(i)}}{\partial \mathbf{w}^{(l)}} = \frac{\partial}{\partial \mathbf{w}^{(l)}} [(\mathbf{x}^{(i)})^T \mathbf{w}^{(l)} + b_l] = \mathbf{x}^{(i)} \in \mathbb{R}^m \quad (\nabla_{\mathbf{w}} [\mathbf{a}^T \mathbf{w}] = \mathbf{a})$$

Case $l = k$

$$\frac{\partial \hat{y}_k^{(i)}}{\partial z_k^{(i)}} = \frac{\partial}{\partial z_k^{(i)}} \left[\frac{\exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})} \right]$$

$$= \frac{\exp(z_k^{(i)}) \cdot \sum_{k'=1}^{c} \exp(z_{k'}^{(i)}) - \exp(z_k^{(i)}) \cdot \exp(z_k^{(i)})}{[\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})]^2} \quad \left(\frac{d}{dx} \left[\frac{f}{g} \right] = \frac{f'g - fg'}{g^2} \right)$$

$$= \frac{\exp(z_k^{(i)}) \cdot [\sum_{k'=1}^{c} \exp(z_{k'}^{(i)}) - \exp(z_k^{(i)})]}{[\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})]^2}$$

$$= \frac{\exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})} \cdot \frac{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)}) - \exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})}$$

$$= \hat{y}_k^{(i)} \cdot \left(1 - \frac{\exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})}\right)$$

$$= \hat{y}_k^{(i)}(1 - \hat{y}_k^{(i)})$$

$$\therefore \nabla_{\mathbf{w}^{(k)}} \hat{y}_k^{(i)} = \mathbf{x}^{(i)} \hat{y}_k^{(i)}(1 - \hat{y}_k^{(i)}) \in \mathbb{R}^m$$

Case $l \neq k$

$$\frac{\partial \hat{y}_k^{(i)}}{\partial z_l^{(i)}} = \frac{\partial}{\partial z_l^{(i)}} \left[\frac{\exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})}\right]$$

$$= \frac{0 \cdot \sum_{k'=1}^{c} \exp(z_{k'}^{(i)}) - \exp(z_k^{(i)}) \cdot \exp(z_l^{(i)})}{[\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})]^2} \quad \left(\frac{\partial \exp(z_k)}{\partial z_l} = 0,\ k \neq l\right)$$

$$= -\frac{\exp(z_k^{(i)}) \cdot \exp(z_l^{(i)})}{[\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})]^2}$$

$$= -\frac{\exp(z_k^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})} \cdot \frac{\exp(z_l^{(i)})}{\sum_{k'=1}^{c} \exp(z_{k'}^{(i)})}$$

$$= -\hat{y}_k^{(i)} \hat{y}_l^{(i)}$$

$$\therefore \nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} = -\mathbf{x}^{(i)} \hat{y}_k^{(i)} \hat{y}_l^{(i)} \in \mathbb{R}^m$$

Step 2 $\nabla_{\mathbf{w}^{(l)}} f_{CE}$

$$\nabla_{\mathbf{w}^{(l)}} f_{CE} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \nabla_{\mathbf{w}^{(l)}} \log \hat{y}_k^{(i)}$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{c} y_k^{(i)} \cdot \frac{1}{\hat{y}_k^{(i)}} \cdot \nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)} \quad \left(\frac{d}{dx}\log f = \frac{1}{f}\cdot\frac{df}{dx}\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\sum_{k=1}^{c} y_k^{(i)} \cdot \frac{\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)}}{\hat{y}_k^{(i)}}\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[y_l^{(i)} \cdot \frac{\nabla_{\mathbf{w}^{(l)}} \hat{y}_l^{(i)}}{\hat{y}_l^{(i)}} + \sum_{k\neq l} y_k^{(i)} \cdot \frac{\nabla_{\mathbf{w}^{(l)}} \hat{y}_k^{(i)}}{\hat{y}_k^{(i)}}\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[y_l^{(i)} \cdot \frac{\mathbf{x}^{(i)}\hat{y}_l^{(i)}(1-\hat{y}_l^{(i)})}{\hat{y}_l^{(i)}} + \sum_{k\neq l} y_k^{(i)} \cdot \frac{-\mathbf{x}^{(i)}\hat{y}_k^{(i)}\hat{y}_l^{(i)}}{\hat{y}_k^{(i)}}\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{x}^{(i)} y_l^{(i)}(1-\hat{y}_l^{(i)}) - \mathbf{x}^{(i)}\hat{y}_l^{(i)}\sum_{k\neq l} y_k^{(i)}\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left[y_l^{(i)} - y_l^{(i)}\hat{y}_l^{(i)} - \hat{y}_l^{(i)}\sum_{k\neq l} y_k^{(i)}\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left[y_l^{(i)} - \hat{y}_l^{(i)}\left(y_l^{(i)} + \sum_{k\neq l} y_k^{(i)}\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left[y_l^{(i)} - \hat{y}_l^{(i)}\sum_{k=1}^{c} y_k^{(i)}\right] \quad \left(\sum_{k=1}^{c} = \sum_{k=l} + \sum_{k\neq l}\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}\left[y_l^{(i)} - \hat{y}_l^{(i)} \cdot 1\right] \quad \left(\sum_{k=1}^{c} y_k^{(i)} = 1 \text{ for one-hot}\right)$$

$$\nabla_{\mathbf{w}^{(l)}} f_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}^{(i)}(y_l^{(i)} - \hat{y}_l^{(i)}) \in \mathbb{R}^m$$

Step 3 $\nabla_{\mathbf{b}} f_{CE}$

$$\frac{\partial z_k^{(i)}}{\partial b_l} = \frac{\partial}{\partial b_l}[(\mathbf{x}^{(i)})^T\mathbf{w}^{(k)} + b_k] = \delta_{kl} = \begin{cases} 1 & k = l \\ 0 & k \neq l \end{cases}$$

$$\frac{\partial \hat{y}_k^{(i)}}{\partial b_l} = \frac{\partial \hat{y}_k^{(i)}}{\partial z_l^{(i)}} \cdot \frac{\partial z_l^{(i)}}{\partial b_l} = \frac{\partial \hat{y}_k^{(i)}}{\partial z_l^{(i)}} \cdot 1$$

$$= \begin{cases} \hat{y}_k^{(i)}(1 - \hat{y}_k^{(i)}) & k = l \\ -\hat{y}_k^{(i)} \hat{y}_l^{(i)} & k \neq l \end{cases}$$

$$\nabla_{b_l} f_{CE} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{c} y_k^{(i)} \cdot \frac{1}{\hat{y}_k^{(i)}} \cdot \frac{\partial \hat{y}_k^{(i)}}{\partial b_l}$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)} \cdot \frac{\hat{y}_l^{(i)}(1 - \hat{y}_l^{(i)})}{\hat{y}_l^{(i)}} + \sum_{k \neq l} y_k^{(i)} \cdot \frac{-\hat{y}_k^{(i)} \hat{y}_l^{(i)}}{\hat{y}_k^{(i)}} \right]$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)}(1 - \hat{y}_l^{(i)}) - \hat{y}_l^{(i)} \sum_{k \neq l} y_k^{(i)} \right]$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)} - y_l^{(i)} \hat{y}_l^{(i)} - \hat{y}_l^{(i)} \sum_{k \neq l} y_k^{(i)} \right]$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)} - \hat{y}_l^{(i)} \left(y_l^{(i)} + \sum_{k \neq l} y_k^{(i)} \right) \right]$$

$$= -\frac{1}{n} \sum_{i=1}^{n} \left[y_l^{(i)} - \hat{y}_l^{(i)} \sum_{k=1}^{c} y_k^{(i)} \right]$$

$$= -\frac{1}{n} \sum_{i=1}^{n} (y_l^{(i)} - \hat{y}_l^{(i)})$$

$$\nabla_{\mathbf{b}} f_{CE} = -\frac{1}{n} \sum_{i=1}^{n} \begin{bmatrix} y_1^{(i)} - \hat{y}_1^{(i)} \\ \vdots \\ y_c^{(i)} - \hat{y}_c^{(i)} \end{bmatrix} = -\frac{1}{n} \sum_{i=1}^{n} (\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}) \in \mathbb{R}^c$$

Problem 2: Numpy implementation of softmax regression [20 points]

Image Placeholder: `![Placeholder for image under Problem 2](image_placeholder.png)`

Train a 2-layer softmax neural network to classify images of fashion items (10 different classes, such as shoes, t-shirts, dresses, etc.) from the Fashion MNIST dataset. The input to the network will be a 28×28-pixel image (converted into a 784-dimensional vector); the output will be a vector of 10 probabilities (one for each class). The cross-entropy loss function that you minimize should be

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

where n is the number of examples and α is a regularization constant. Note that each $\hat{y}_k$ implicitly depends on all the weights

$$W = \left[w^{(1)}, \ldots, w^{(10)}\right]$$

and biases

$$b = \left[b^{(1)}, \ldots, b^{(10)}\right].$$

To get started, first download the Fashion MNIST dataset 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

To save some time, you can use this STARTER CODE: https://canvas.wpi.edu/courses/76771/files/folder/assignments?preview=7827760.

These files can be loaded into numpy using `np.load`. Each "labels" file consists of a 1-d array containing n labels (valued 0–9), and each "images" file contains a 2-d array of size $n \times 784$, where n is the number of images. *Hint:* Images are in grayscale and represented by values between 0 and 255. Mapping the inputs to the interval $[-0.5, 0.5]$ will help train the neural network.

Next, implement stochastic gradient descent (SGD) to minimize the cross-entropy loss function on this dataset. Regularize the weights but not the biases. Optimize the hyperparameters (e.g., learning rate, regularization strength), considering at least 9 combinations of hyperparameter values. You should also use the same methodology as for the previous homework, including the splitting of the training files into validation and training portions.

Performance evaluation:

Once you have tuned the hyperparameters and optimized the weights so as to maximize performance on the validation set, then:

1. stop training the network and
2. evaluate the network on the test set.

Record the performance both in terms of (unregularized) cross-entropy loss (smaller is better) and percent correctly classified examples (larger is better); put this information into the PDF you submit.

Answer:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

##### Edit directory for NH/TBA #####
import os
os.chdir("/Users/tomrannosaurus/Downloads/HW2_DEEPLEARNING_NH/")
################################
```

```
def to_one_hot(labels):
    """Convert integer labels to one-hot encoding; returns one-hot array."""
    num_classes = len(set(labels))
    one_hot = np.zeros((len(labels), num_classes))
    one_hot[np.arange(len(labels)), labels - 1] = 1
    return one_hot

def softmax(z):
    """Compute row-wise softmax; returns probability array."""
    # BEGIN YOUR CODE HERE (~1-3 lines)


    z = z- z.max(axis=1, keepdims=True)
    z = np.exp(z)
    proba = z /z.sum(axis=1, keepdims =True)

    # END YOUR CODE HERE
    return proba


def cross_entropy_loss(W, b, images, labels, alpha):
    """Compute cross-entropy loss for predictions; returns scalar loss."""
    # BEGIN YOUR CODE HERE (~2-6 lines)
    logits =  images@W + b
    proba = softmax(logits)

    class_index = np.argmax(labels,axis=1)
    prob_true = proba[np.arange(images.shape[0]), class_index]


    if np.any(prob_true == 0):
        prob_true = 1e-6
        loss = -np.mean(np.log(prob_true))

    else:
        loss = -np.mean(np.log(prob_true))


    loss += 0.5 * alpha * np.sum(W ** 2) # here L2 reg on the weights only


    # END YOUR CODE HERE



    return loss

def compute_gradient(W, b, images, labels, alpha):
    """Compute gradients w.r.t. weights and bias; returns (dW, db)."""
    # BEGIN YOUR CODE HERE (~4-7 lines)
    logits =  images@W + b
    proba = softmax(logits)

    B=images.shape[0]
    Delta = (proba - labels)/B

    dW = images.T @ Delta
    dW += alpha * W # this applies L2 in the gradient

    db = Delta.sum(axis=0)

    # END YOUR CODE HERE
    return dW, db

def compute_accuracy(W, b, images, labels):
    """Compute classification accuracy; returns fraction correct."""
    # BEGIN YOUR CODE HERE (~2-5 lines)

    logits =  images@W + b
    proba = softmax(logits)

    prediction = np.argmax(proba, axis=1)
    output = np.argmax(labels,axis=1)

    acc = np.mean(prediction == output)
    # END YOUR CODE HERE
    return acc

def show_weights(W):
    """Render weights as image patches; returns None."""
    img_size = int(W.shape[0] ** 0.5)
    canvas = np.zeros((img_size, img_size * W.shape[1]))
    for idx, col in enumerate(range(0, canvas.shape[1], img_size)):
        canvas[:, col:col+img_size] = np.reshape(W[:-1, idx], (img_size,
 ↪  img_size))
    plt.imshow(canvas, cmap='gray')
    plt.show()
```

```
def train_softmax_classifier(train_images, train_labels, val_images,
 ↪  val_labels,
                             learning_rate=1e-5, batch_size=16, nepochs=100,
 ↪  alpha=0.0):
    """Train softmax classifier; returns (W, b, loss, accuracy)."""
    num_batches = train_images.shape[0] // batch_size
    n_features = train_images.shape[1]
    n_classes = train_labels.shape[1]

    # Initialize weights and bias
    # BEGIN YOUR CODE HERE (~2 lines)
    W = 0.01 * np.random.randn(n_features, n_classes)
    b = np.zeros(n_classes)
    # END YOUR CODE HERE

    cost = float('inf')
    acc = 0.0
    best_val_loss = float('inf')
    patience = 10
    patience_counter = 0
    best_W_es = 0
    best_b_es = 0
    best_epoch = 0

    for epoch in range(nepochs):
        np.random.seed(epoch)
        perm = np.random.permutation(train_images.shape[0])
        train_images = train_images[perm]
        train_labels = train_labels[perm]
        for batch in range(num_batches):
            # what do you need to do in each iteration?
            # BEGIN YOUR CODE HERE (~5 lines)

            s = batch * batch_size
            e = s + batch_size

            yb = train_labels[s:e]
            xb = train_images[s:e]



            dW, db = compute_gradient(W, b, xb, yb, alpha)

            W -= learning_rate * dW
            b -= learning_rate * db
            # END YOUR CODE HERE
        # what do you need to compute at the end of each epoch?
        # BEGIN YOUR CODE HERE (~3 lines)
        cost = cross_entropy_loss(W, b, val_images, val_labels, alpha)
        acc  = compute_accuracy(W, b, val_images, val_labels)
        # END YOUR CODE HERE
        if cost < best_val_loss:
            best_val_loss = cost
            best_W_es = W.copy()
            best_b_es = b.copy()
            best_epoch = epoch + 1
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                break

    if best_W_es is not None:
        return best_W_es, best_b_es, best_val_loss,
         ↪  compute_accuracy(best_W_es, best_b_es, val_images, val_labels),
         ↪  best_epoch
    return W, b, cost, acc, epoch + 1

# def add_bias(images):
#     return np.hstack((images, np.ones((images.shape[0], 1))))

if __name__ == "__main__":
    # Load data
    np.random.seed(541)

    training_images = np.load("fashion_mnist_train_images.npy") / 255.0 - 0.5
    training_labels = to_one_hot(np.load("fashion_mnist_train_labels.npy"))
    testing_images = np.load("fashion_mnist_test_images.npy") / 255.0 - 0.5
    testing_labels = to_one_hot(np.load("fashion_mnist_test_labels.npy"))

    # split training into training and validation using train_test_split
    x_train, x_val, y_train, y_val = train_test_split(training_images,
 ↪  training_labels,
                                                      test_size=0.2,
 ↪  random_state=541)
```

```
    # List hyperparameters to try
    # BEGIN YOUR CODE HERE (~2-3 lines)
    batch_sizes_list = [8,16,32,64]

    learning_rates_list =[0.1,0.01,0.001]

    alpha_values = [0.0, 0.001, 0.01]
    # END YOUR CODE HERE

    # Initialize varjables to keep track of best hyperparameters
    # BEGIN YOUR CODE HERE (~3 lines)
    best_learning_rate = 0
    best_bs = 0
    best_accuracy = -1.0
    W_best = 0
    b_best = 0
    best_alpha = 0


    # END YOUR CODE HERE

    # Train model
    # BEGIN YOUR CODE HERE (~7-10 lines)

    # Train model (grid over all unique combos)
    run_idx = 0
    total = len(learning_rates_list) * len(batch_sizes_list) *
↪   len(alpha_values)

    for alpha in alpha_values:
        for bts in batch_sizes_list:
            for lnr in learning_rates_list:

                run_idx += 1
                print(f"Starting combo {run_idx}/{total}: lr={lnr},
                 ↪  batch_size={bts}, alpha={alpha}", flush=True)

                w, bias, loss, accuracy, epochs_used =
↪   train_softmax_classifier(
                    x_train, y_train, x_val, y_val,
                    learning_rate=lnr, batch_size=bts, nepochs=1000,
↪   alpha=alpha
```

```
                )

                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    best_learning_rate = lnr
                    best_bs = bts
                    best_alpha = alpha
                    best_epochs_used = epochs_used
                    W_best = w
                    b_best = bias
                    print(f" -> new best acc={best_accuracy:.4f}",
                     ↪  flush=True)

print("Tested all unique hyperparameter combinations.")

    # END YOUR CODE HERE

# Retrain model on full training set with best hyperparameters and evaluate
 ↪  on test set
# BEGIN YOUR CODE HERE (~1 line)
final_W, final_b, loss, acc, final_epochs = train_softmax_classifier(
    training_images, training_labels, testing_images, testing_labels,
    learning_rate=best_learning_rate, batch_size=best_bs, nepochs=1000,
 ↪  alpha=best_alpha)

test_loss = cross_entropy_loss(final_W, final_b, testing_images,
 ↪  testing_labels, best_alpha)
test_accuracy = compute_accuracy(final_W, final_b, testing_images,
 ↪  testing_labels)

np.savez('hw2_p2_results.npz',
        final_W=final_W, final_b=final_b, W_best=W_best, b_best=b_best,
        best_learning_rate=best_learning_rate, best_bs=best_bs,
 ↪  best_alpha=best_alpha,
        test_loss=test_loss, test_accuracy=test_accuracy,
        val_accuracy=best_accuracy, best_epochs_used=best_epochs_used)
```

```
#load
loaded = np.load('hw2_p2_results.npz')
```

```
# print
print("Best Hyperparameters Found:")
print(f" Learning Rate: {loaded['best_learning_rate']}")
print(f" Batch Size: {loaded['best_bs']}")
print(f" Regularization (alpha): {loaded['best_alpha']}")
print(f" Best Epoch: {loaded['best_epochs_used']}")

print("Test Set Performance:")
print(f" Cross-Entropy Loss (unregularized): {loaded['test_loss']:.4f}")
print(f" Accuracy: {loaded['test_accuracy']:.4f}
 ↪   ({loaded['test_accuracy']*100:.2f}%)")

# END YOUR CODE HERE
```

```
Best Hyperparameters Found:
 Learning Rate: 0.01
 Batch Size: 32
 Regularization (alpha): 0.0
 Best Epoch: 115
Test Set Performance:
 Cross-Entropy Loss (unregularized): 0.4350
 Accuracy: 0.8452 (84.52%)
```

Problem 3: Instrumenting and Tracking Experiments with Weights & Biases [10 points]

In Problem 2, you implemented softmax regression and manually tuned its hyperparameters by running your code multiple times with different settings. You likely discovered that this process can be challenging. How did you keep track of which learning rate and regularization strength produced which validation accuracy? Was it easy to compare the training dynamics of run #3 with run #8? How could you share your findings with a collaborator to defend your choice of the final model?

In academic research and industry, this process of tracking, comparing, and managing experiments is a core component of Machine Learning Operations (MLOps). It is managed using specialized platforms designed to make machine learning a systematic and reproducible science. In this problem, you will augment your solution from Problem 2 using one of the most popular experiment tracking platforms: **Weights & Biases (W&B)**. Your goal is not just to run experiments, but to log them methodically and use the W&B dashboard to analyze your results in a way that was not possible with your previous manual approach.

This problem is based on the steps described in this tutorial.

Part A: Setup and W&B Integration [4 points]

Your first task is to set up your environment and instrument your existing script from Problem 2 to communicate with the W&B platform.

1. Create a free personal account at https://wandb.ai/site.
2. Install the W&B library in your Python environment using a package manager: `pip install wandb`.
3. (Recommended, but optional step) Authenticate your machine by running the following command in your terminal: `wandb login`. You will be prompted to paste an API key, which you can find in the ”Settings” section of your W&B profile. *Note:* if you don't log in using the terminal, you will be prompted to insert your API key when first running your code.
4. Copy your script from Problem 2 to create a modified version. At the beginning of your new script, after your imports, add `import wandb`.
5. (1 point) Structure your code to execute the 9+ hyperparameter combinations from Problem 2 within a loop. For each unique combination of hyperparameters, you will perform the steps below to initialize and track a distinct experiment run.
6. (1 point) Inside this loop, before you begin training, use the `wandb.init()` function to start a new run. This function is the entry point for all tracking. You must configure it as follows:
7. (1 point) Set the `project` argument to a single, descriptive name for all runs in this assignment, for example, `"cs541-hw1-fashion-mnist"`. All your runs will be grouped under this project on your dashboard.
8. (1 point) Pass a Python dictionary containing the hyperparameters for the current run to the `config` argument. This is crucial for telling W&B which settings correspond to which results. For example:

   ```
   {'learning rate': 0.01, 'alpha': 1e-4, 'epochs': 20}
   ```

9. To ensure your code uses the exact hyperparameters logged by W&B, it is a best practice to access them throughout your training logic via the run's `config` object. For example, after initialization, use

   ```
   learning_rate = wandb.config.learning_rate
   ```

   instead of your original variable. This establishes the W&B configuration as the single source of truth for the experiment, a key principle of reproducibility.

Answer

```
import wandb

def train_softmax_clssfr_wandb(train_images, train_labels, val_images,
 ↪  val_labels,
                              learning_rate=1e-5, batch_size=16, nepochs=100,
 ↪  alpha=0.0, log_to_wandb=True):
    """Train softmax classifier; returns (W, b, loss, accuracy)."""

    if log_to_wandb and wandb.run is not None:
        learning_rate = wandb.config.learning_rate
        batch_size = wandb.config.batch_size
        alpha = wandb.config.alpha
        nepochs = wandb.config.epochs

    num_batches = train_images.shape[0] // batch_size
    n_features = train_images.shape[1]
    n_classes = train_labels.shape[1]

    # Initialize weights and bias
    # BEGIN YOUR CODE HERE (~2 lines)
    W = 0.01 * np.random.randn(n_features, n_classes)
    b = np.zeros(n_classes)
    # END YOUR CODE HERE

    cost = float('inf')
    acc = 0.0
    best_val_loss = float('inf')
    patience = 10
    patience_counter = 0
    best_W_es = 0
    best_b_es = 0
    best_epoch = 0

    for epoch in range(nepochs):
        np.random.seed(epoch)
        perm = np.random.permutation(train_images.shape[0])
        train_images = train_images[perm]
        train_labels = train_labels[perm]
        for batch in range(num_batches):
            # what do you need to do in each iteration?
            # BEGIN YOUR CODE HERE (~5 lines)

            s = batch * batch_size
            e = s + batch_size

            yb = train_labels[s:e]
            xb = train_images[s:e]

            dW, db = compute_gradient(W, b, xb, yb, alpha)

            W -= learning_rate * dW
            b -= learning_rate * db

            # END YOUR CODE HERE
        # what do you need to compute at the end of each epoch?
        # BEGIN YOUR CODE HERE (~3 lines)
        cost = cross_entropy_loss(W, b, val_images, val_labels, alpha)
        acc = compute_accuracy(W, b, val_images, val_labels)
        # END YOUR CODE HERE

        # log to wandb
        if log_to_wandb and wandb.run is not None:
            wandb.log({
                "epoch": epoch + 1,
                "val_loss": cost,
                "val_accuracy": acc})

        if cost < best_val_loss:
            best_val_loss = cost
            best_W_es = W.copy()
            best_b_es = b.copy()
            best_epoch = epoch + 1
            patience_counter = 0
        else:
            patience_counter += 1
            if patience_counter >= patience:
                break

    if best_W_es is not None:
        return best_W_es, best_b_es, best_val_loss,
         ↪  compute_accuracy(best_W_es, best_b_es, val_images, val_labels),
         ↪  best_epoch
    return W, b, cost, acc, epoch + 1
```

Part B: Logging Metrics During Training [2 points]

With initialization in place, you must now log the model's performance metrics to W&B throughout the training process. This allows you to capture the dynamics of learning, not just the final result.

1. Locate the training loop (the one that iterates over epochs) in your script.
2. Within this loop, immediately after you evaluate your model on the validation set at the end of an epoch, add a call to the `wandb.log()` function.
3. As described in the W&B documentation, this function accepts a dictionary of metrics. Your dictionary must contain, at a minimum, the validation loss and validation accuracy for the current epoch. It is also good practice to log the epoch number itself.
4. (1 point) A correct log call should look similar to this:

   ```
   wandb.log({'epoch': epoch, 'val_loss': validation_loss, 'val_accuracy':
    ↪  validation_accuracy})
   ```

5. (1 point) Ensure this line is executed for every epoch of every hyperparameter combination run. W&B will automatically aggregate this data to generate time-series plots for each metric, allowing you to visualize learning curves for every experiment.

Answer

```
if __name__ == "__main__":
    # Load data
    np.random.seed(541)

    training_images = np.load("fashion_mnist_train_images.npy") / 255.0 - 0.5
    training_labels = to_one_hot(np.load("fashion_mnist_train_labels.npy"))
    testing_images = np.load("fashion_mnist_test_images.npy") / 255.0 - 0.5
    testing_labels = to_one_hot(np.load("fashion_mnist_test_labels.npy"))
```

```
# split training into training and validation using train_test_split
x_train, x_val, y_train, y_val = train_test_split(training_images, training_labels, test_size=0.2, random_state=541)

# List hyperparameters to try
# BEGIN YOUR CODE HERE (~2-3 lines)
batch_sizes_list = [8,16,32,64]

learning_rates_list =[0.1,0.01,0.001]

alpha_values = [0.0, 0.001, 0.01]
# END YOUR CODE HERE

# Initialize varjables to keep track of best hyperparameters
# BEGIN YOUR CODE HERE (~3 lines)
best_learning_rate = 0
best_bs = 0
best_accuracy = -1.0
W_best = 0
b_best = 0
best_alpha = 0


# END YOUR CODE HERE

# Train model
# BEGIN YOUR CODE HERE (~7-10 lines)

# Train model (grid over all unique combos)
run_idx = 0
total = len(learning_rates_list) * len(batch_sizes_list) * len(alpha_values)

for alpha in alpha_values:
    for bts in batch_sizes_list:
        for lnr in learning_rates_list:

            run_idx += 1
            print(f"Starting combo {run_idx}/{total}: lr={lnr}, batch_size={bts}, alpha={alpha}", flush=True)
```

```
                # wandb init
                wandb.init(
                    project="cs541-hw1-fashion-mnist",
                    name=f"run_{run_idx}_lr{lnr}_bs{bts}_a{alpha}",
                    config={
                        "learning_rate": lnr,
                        "batch_size": bts,
                        "alpha": alpha,
                        "epochs": 1000
                    },
                    reinit=True
                )

                #run training
                w, bias, loss, accuracy, epochs_used = train_softmax_clssfr_wandb(
                  x_train, y_train, x_val, y_val,
                  learning_rate=wandb.config.learning_rate,
                  batch_size=wandb.config.batch_size,
                  nepochs=wandb.config.epochs,
                  alpha=wandb.config.alpha,
                  log_to_wandb=True
               )

                # log to wandb
                wandb.log({
                  "final_val_loss": loss,
                  "final_val_accuracy": accuracy,
                  "epochs_trained": epochs_used
                })

                if accuracy > best_accuracy:
                  best_accuracy = accuracy
                  best_learning_rate = lnr
                  best_bs = bts
                  best_alpha = alpha
                  best_epochs_used = epochs_used
                  W_best = w
                  b_best = bias
                  print(f" -> new best acc={best_accuracy:.4f}", flush=True)

                wandb.finish()
```

```
print("Tested all unique hyperparameter combinations.")

# rerun best on full training set and test for final results (same as above
 ↪  but now with wandb logging)
wandb.init(
   project="cs541-hw1-fashion-mnist",
   name="best_model_final_training",
   config={
      "learning_rate": best_lr,
      "batch_size": best_bs,
      "alpha": best_alpha,
      "epochs": 1000,
      "type": "final_training"
   },
   reinit=True
)

final_W, final_b, loss, acc, final_epochs =
 ↪  train_softmax_classifier_with_wandb(
   training_images, training_labels, testing_images, testing_labels,
   learning_rate=wandb.config.learning_rate,
   batch_size=wandb.config.batch_size,
   nepochs=wandb.config.epochs,
   alpha=wandb.config.alpha,
   log_to_wandb=True
)

test_loss = cross_entropy_loss(final_W, final_b, testing_images,
 ↪  testing_labels, best_alpha)
test_accuracy = compute_accuracy(final_W, final_b, testing_images,
 ↪  testing_labels)

wandb.log({
   "test_loss": test_loss,
   "test_accuracy": test_accuracy})

wandb.finish()
```

Part C: Analysis and Visualization [4 points]

After running all your hyperparameter experiments and logging them to W&B, the final and most important step is to use the platform's tools to analyze your results.

1. Navigate to your project page on the W&B website. Under the tab "Runs", you should see a table summarizing all the runs you executed, with columns for each of your configured hyperparameters and logged metrics.
2. Your task is to analyze the relationship between your chosen hyperparameters and the model's performance. In the W&B user interface, create a **Parallel Coordinates Plot**. This plot is a powerful tool for visualizing how different hyperparameter values correlate with output metrics. Configure the plot to show your hyperparameters (e.g., learning rate, alpha) as axes, along with your primary performance metric (e.g., the final validation accuracy).
3. (1 point) Take a screenshot of your configured Parallel Coordinates Plot and include it in your submitted PDF report.
4. (3 points) Below the screenshot, write a brief analysis (2–3 paragraphs) that answers the following questions:
   - Based on the visualization, which hyperparameter had the most significant impact on validation accuracy? How does the plot make this apparent?
   - What range of values for each hyperparameter appears to be most promising for achieving high performance?
   - How does this visualization-driven analysis make it easier to draw these conclusions compared to looking at a simple table of numbers or your console output from Problem 2? What are the key advantages of this systematic approach?

Answer

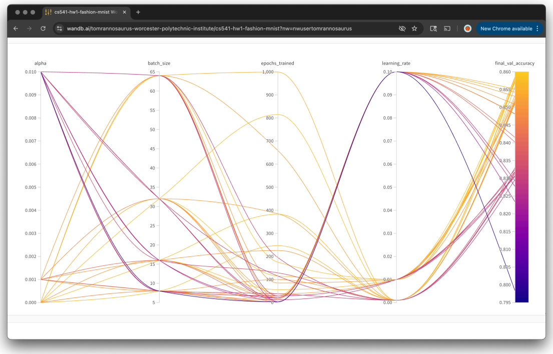

Figure 1: Parallel Coordinates Plot: All Runs

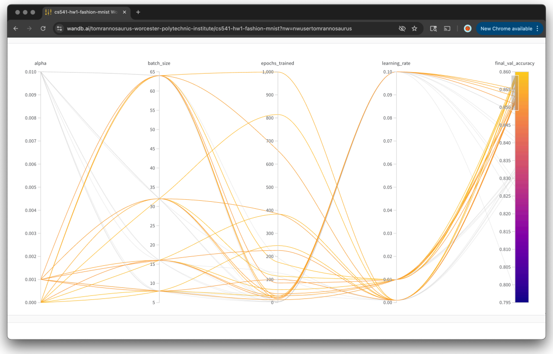

Figure 3: Parallel Coordinates Plot: Filtered Top Runs 2

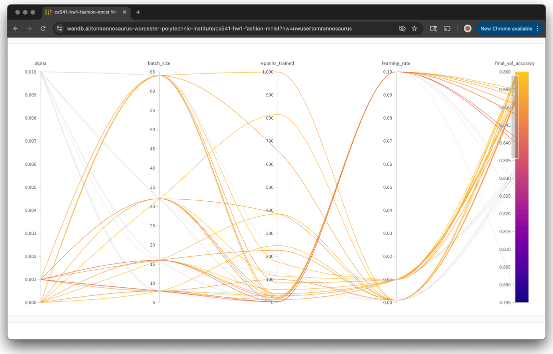

Figure 2: Parallel Coordinates Plot: Filtered Top Runs 1

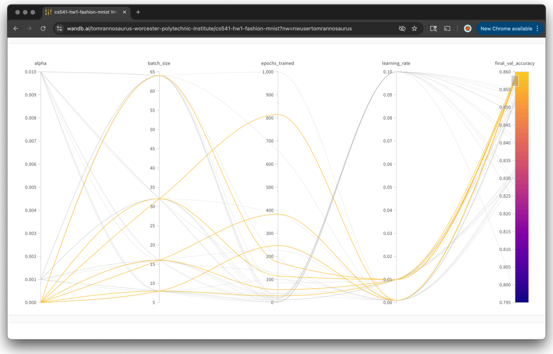

Figure 4: Parallel Coordinates Plot: Filtered Top Runs 3

Based on the parallel coordinates visualization, learning rate had the most significant impact on validation accuracy. This is evident from the performance stratification where all poor-performing runs converge through learning rate 0.10, while high-performing runs pass through 0.01. Learning rates of 0.01 and sometimes 0.001 produced optimal results; 0.10 consistently failed, likely due to optimization instability. Regularization parameter alpha showed secondary effects, with 0.010 correlating with decreased performance versus 0.000-0.001. Batch size had minimal impact, with successful runs distributed across all values.

The visualization provides clear advantages over tabular output. Pattern recognition is immediate; the problematic learning rate 0.10 range would be difficult to spot scanning 36 rows of numbers. Interaction effects become visible, we see how learning rate 0.10 consistently results in the best final accuracy values, regardless of other settings. The plot gives a unified view of the hyperparameter space.

Problem 4: Implementing gradient computation via the multivariate chain rule (linear case) [15 points]

Consider the two vector fields below taken from the Supplemental Materials on Jacobian matrices and multivariate chain rule, slide 9:

$$f\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = x_1 - 2x_2 + \tfrac{x_3}{4}$$

$$g\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 + 2x_2 \\ x_2 \\ -x_1 + 3 \end{bmatrix}$$

Let the initial input to g be

$$x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

---

1. Show how to represent each function as an affine transformation of the form $Wx + b$. For function g, denote the parameters by W_1 and b_1, whereas for function f, denote the parameters by W_2 and b_2. [1 point]
2. Implement a python function `AffineTransformation(W, b, x)` that can be used to compute either function by taking as input two arrays of parameters W and b and a vector x. [1 point]
3. Building on the previous function, implement a new function `Composition(L, x)` that calculates the composition of affine transformations represented as a list of pairs

   $$L = [(W_1, b_1), \dots, (W_n, b_n)]$$

   and an initial vector x. Use the following convention: the functions are applied from the smallest to the largest index. Return all the activation values (i.e., intermediate results) as a list variable

$$z = (z_1, \dots, z_n).$$

[2 points]

4. How would you call the previous function to compute $(f \circ g)(x)$, for $x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$? Return **ONLY** the final output. [1 point]
5. Given an affine transformation $z_2 = f(z_1)$ parameterized by W_2 and b_2, derive [2 points]:

$$\frac{\partial z_2}{\partial \operatorname{vec}[W_2]}(z_1) \quad \text{and} \quad \frac{\partial z_2}{\partial b_2}(z_1).$$

6. Write equations for: [1 point]

$$\frac{\partial (f \circ g)}{\partial z_1}(x)$$

7. For some scalar function $y = (f \circ g)(x)$, where $z = g(x)$ is an affine transformation parameterized by $W_{n \times m}$ and $b_{n \times 1}$, we know that

$$\nabla_W y = \operatorname{unvec}\left[\frac{\partial y}{\partial z} \frac{\partial z}{\partial \operatorname{vec}[W]}\right]$$

Last, if

$$\frac{\partial y}{\partial z} = p^\top,$$

then

$$\nabla_W y = p x^\top.$$

Using the multivariate chain rule, write equations for: [2 points]

$$\frac{\partial (f \circ g)}{\partial b_1} \quad \text{and} \quad \nabla_{W_1} (f \circ g)(x)$$

8. Using the list of parameters L, the input vector x and all the activation values z obtained from the Composition, implement a function `ComputeGradients(L,x,z)` that returns the gradients of z_n (a scalar) for all the parameters in L, i.e.,

$$\nabla_{b_i} z_n = \left(\frac{\partial z_n}{\partial b_i}\right)^\top, \quad \nabla_{W_i} z_n.$$

To earn full marks, the solution should apply to any $n \geq 2$. [5 points]

Answers

1.

Function g ($\mathbb{R}^2 \to \mathbb{R}^3$)

$$g\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}x_1+2x_2\\x_2\\-x_1+3\end{pmatrix} = \begin{pmatrix}1\cdot x_1 + 2\cdot x_2 + 0\\0\cdot x_1 + 1\cdot x_2 + 0\\(-1)\cdot x_1 + 0\cdot x_2 + 3\end{pmatrix}$$

$$= \begin{pmatrix}1 & 2\\0 & 1\\-1 & 0\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}0\\0\\3\end{pmatrix}$$

$$\mathbf{W_1} = \begin{pmatrix}1 & 2\\0 & 1\\-1 & 0\end{pmatrix} \in \mathbb{R}^{3\times 2}, \quad \mathbf{b_1} = \begin{pmatrix}0\\0\\3\end{pmatrix} \in \mathbb{R}^3$$

Function f: ($\mathbb{R}^3 \to \mathbb{R}^1$)

$$f\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} = x_1 - 2x_2 + \frac{x_3}{4} = \begin{pmatrix}1 & -2 & \frac{1}{4}\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} + 0$$

$$\mathbf{W_2} = \begin{pmatrix}1 & -2 & \frac{1}{4}\end{pmatrix} \in \mathbb{R}^{1\times 3}, \quad b_2 = 0 \in \mathbb{R}$$

2.

```python
def AffineTransformation(W, b, x):
    return W @ x + b
```

3.

```python
def Composition(L, x):
    z = [x]
    for W_i, b_i in L:
        z.append(W_i @ z[-1] + b_i)
    return z[1:]
```

4.

$$(f \circ g)(\mathbf{x}) \text{ for } \mathbf{x} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$\mathbf{z_1} = g(\mathbf{x}) = \mathbf{W_1}\mathbf{x} + \mathbf{b_1}$$

$$= \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix}$$

$$= \begin{pmatrix} (1)(-1) + (2)(1) \\ (0)(-1) + (1)(1) \\ (-1)(-1) + (0)(1) \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 4 \end{pmatrix}$$

$$z_2 = f(\mathbf{z_1}) = \mathbf{W_2}\mathbf{z_1} + b_2$$

$$= \begin{pmatrix} 1 & -2 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 4 \end{pmatrix} + 0 = (1)(1) + (-2)(1) + \left(\frac{1}{4}\right)(4) = 0$$

5.

(a) $\frac{\partial z_2}{\partial \mathrm{vec}[\mathbf{W_2}]}(\mathbf{z_1})$

$$\mathrm{vec}_{\mathrm{row}}[\mathbf{W_2}] = \begin{pmatrix} W_{2,1} \\ W_{2,2} \\ W_{2,3} \end{pmatrix} \in \mathbb{R}^3 \quad \text{(row-wise vec for } 1 \times 3 \text{ matrix)}$$

$$z_2 = W_{2,1}z_{1,1} + W_{2,2}z_{1,2} + W_{2,3}z_{1,3} + b_2$$

$$\frac{\partial z_2}{\partial W_{2,j}} = \frac{\partial}{\partial W_{2,j}} \left[\sum_{k=1}^{3} W_{2,k}z_{1,k} + b_2 \right] = z_{1,j}$$

$$\frac{\partial z_2}{\partial \mathrm{vec}[\mathbf{W_2}]} = \begin{pmatrix} z_{1,1} & z_{1,2} & z_{1,3} \end{pmatrix} = \mathbf{z_1}^T \in \mathbb{R}^{1\times 3}$$

(b) $\frac{\partial z_2}{\partial b_2}(\mathbf{z_1})$

$$\frac{\partial z_2}{\partial b_2} = \frac{\partial}{\partial b_2}[\mathbf{W_2}\mathbf{z_1} + b_2] = 1$$

6.

$\frac{\partial (f\circ g)}{\partial \mathbf{z_1}}(\mathbf{x})$:

$$\frac{\partial z_2}{\partial z_{1,i}} = \frac{\partial}{\partial z_{1,i}} \left[\sum_{j=1}^{3} W_{2,j}z_{1,j} + b_2 \right] = W_{2,i}$$

$$\frac{\partial (f \circ g)}{\partial \mathbf{z_1}} = \begin{pmatrix} W_{2,1} \\ W_{2,2} \\ W_{2,3} \end{pmatrix} = \mathbf{W_2}^T = \begin{pmatrix} 1 \\ -2 \\ \frac{1}{4} \end{pmatrix} \in \mathbb{R}^{3\times 1}$$

7.

(a) $\frac{\partial (f\circ g)}{\partial \mathbf{b_1}}$

$$\mathbf{z_1} = \mathbf{W_1}\mathbf{x} + \mathbf{b_1} \quad \rightarrow \quad \frac{\partial z_{1,k}}{\partial b_{1,i}} = \delta_{ki}$$

$$\frac{\partial z_2}{\partial b_{1,i}} = \sum_{k=1}^{3} \frac{\partial z_2}{\partial z_{1,k}} \cdot \frac{\partial z_{1,k}}{\partial b_{1,i}} = \sum_{k=1}^{3} W_{2,k} \cdot \delta_{ki} = W_{2,i}$$

$$\frac{\partial (f \circ g)}{\partial \mathbf{b_1}} = \begin{pmatrix} W_{2,1} \\ W_{2,2} \\ W_{2,3} \end{pmatrix} = \mathbf{W_2}^T = \begin{pmatrix} 1 \\ -2 \\ \frac{1}{4} \end{pmatrix} \in \mathbb{R}^{3\times 1}$$

(b) $\nabla_{\mathbf{W_1}}(f \circ g)(\mathbf{x})$

$$\text{vec}_{\text{row}}[\mathbf{W_1}] = \begin{pmatrix} W_{11} \\ W_{12} \\ W_{21} \\ W_{22} \\ W_{31} \\ W_{32} \end{pmatrix} \in \mathbb{R}^6 \quad \text{(stack rows sequentially)}$$

$$\mathbf{z_1} = \mathbf{W_1}\mathbf{x} + \mathbf{b_1} = \begin{pmatrix} W_{11}x_1 + W_{12}x_2 + b_{1,1} \\ W_{21}x_1 + W_{22}x_2 + b_{1,2} \\ W_{31}x_1 + W_{32}x_2 + b_{1,3} \end{pmatrix}$$

$$\frac{\partial z_{1,i}}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \left[\sum_{k=1}^{2} W_{ik}x_k + b_{1,i} \right] = x_j$$

$$\frac{\partial \mathbf{z_1}}{\partial \text{vec}_{\text{row}}[\mathbf{W_1}]} = \begin{pmatrix} \frac{\partial z_{1,1}}{\partial W_{11}} & \frac{\partial z_{1,1}}{\partial W_{12}} & \frac{\partial z_{1,1}}{\partial W_{21}} & \frac{\partial z_{1,1}}{\partial W_{22}} & \frac{\partial z_{1,1}}{\partial W_{31}} & \frac{\partial z_{1,1}}{\partial W_{32}} \\ \frac{\partial z_{1,2}}{\partial W_{11}} & \frac{\partial z_{1,2}}{\partial W_{12}} & \frac{\partial z_{1,2}}{\partial W_{21}} & \frac{\partial z_{1,2}}{\partial W_{22}} & \frac{\partial z_{1,2}}{\partial W_{31}} & \frac{\partial z_{1,2}}{\partial W_{32}} \\ \frac{\partial z_{1,3}}{\partial W_{11}} & \frac{\partial z_{1,3}}{\partial W_{12}} & \frac{\partial z_{1,3}}{\partial W_{21}} & \frac{\partial z_{1,3}}{\partial W_{22}} & \frac{\partial z_{1,3}}{\partial W_{31}} & \frac{\partial z_{1,3}}{\partial W_{32}} \end{pmatrix}$$

$$= \begin{pmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{pmatrix} \in \mathbb{R}^{3\times 6}$$

$$\frac{\partial z_2}{\partial \text{vec}_{\text{row}}[\mathbf{W_1}]} = \frac{\partial z_2}{\partial \mathbf{z_1}} \cdot \frac{\partial \mathbf{z_1}}{\partial \text{vec}_{\text{row}}[\mathbf{W_1}]} \quad \text{(chain rule)}$$

$$= \mathbf{W_2} \begin{pmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{pmatrix}$$

$$= \begin{pmatrix} W_{2,1} & W_{2,2} & W_{2,3} \end{pmatrix} \begin{pmatrix} x_1 & x_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & x_1 & x_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 \end{pmatrix}$$

$$= \begin{pmatrix} W_{2,1}x_1 & W_{2,1}x_2 & W_{2,2}x_1 & W_{2,2}x_2 & W_{2,3}x_1 & W_{2,3}x_2 \end{pmatrix}$$

$$\nabla_{\mathbf{W_1}}(f \circ g) = \text{unvec}_{\text{row}}\left[\begin{pmatrix} W_{2,1}x_1 \\ W_{2,1}x_2 \\ W_{2,2}x_1 \\ W_{2,2}x_2 \\ W_{2,3}x_1 \\ W_{2,3}x_2 \end{pmatrix}\right]_{3\times 2}$$

$$= \begin{pmatrix} W_{2,1}x_1 & W_{2,1}x_2 \\ W_{2,2}x_1 & W_{2,2}x_2 \\ W_{2,3}x_1 & W_{2,3}x_2 \end{pmatrix} = \begin{pmatrix} W_{2,1} \\ W_{2,2} \\ W_{2,3} \end{pmatrix} \begin{pmatrix} x_1 & x_2 \end{pmatrix} = \mathbf{W_2}^T \mathbf{x}^T \in \mathbb{R}^{3\times 2}$$

Given Identity:

$$\text{If } \frac{\partial y}{\partial \mathbf{z}} = \mathbf{p}^T \text{ then } \nabla_{\mathbf{W}} y = \mathbf{p}\mathbf{x}^T \quad \text{(outer product)}$$

8.

$$\text{Given: } L = [(\mathbf{W_1}, \mathbf{b_1}), \dots, (\mathbf{W_n}, \mathbf{b_n})], \quad \mathbf{x}, \quad \mathbf{z} = [\mathbf{z_1}, \dots, \mathbf{z_n}]$$

$$\text{Forward pass: } \mathbf{z_0} = \mathbf{x}, \quad \mathbf{z_i} = \mathbf{W_i}\mathbf{z_{i-1}} + \mathbf{b_i}, \quad i = 1, \dots, n$$

$$\text{Initialize: } \frac{\partial z_n}{\partial z_n} = 1 \quad \text{(scalar output)}$$

$$\text{Backward recursion: } \frac{\partial z_n}{\partial \mathbf{z_{i-1}}} = \frac{\partial z_n}{\partial \mathbf{z_i}} \cdot \frac{\partial \mathbf{z_i}}{\partial \mathbf{z_{i-1}}} = \frac{\partial z_n}{\partial \mathbf{z_i}} \cdot \mathbf{W_i}^T$$

$$\nabla_{\mathbf{W_i}} z_n = \left(\frac{\partial z_n}{\partial \mathbf{z_i}}\right) \mathbf{z_{i-1}}^T \quad \text{(apply key identity with } \mathbf{p} = \frac{\partial z_n}{\partial \mathbf{z_i}},\ \mathbf{x} = \mathbf{z_{i-1}}\text{)}$$

$$\nabla_{\mathbf{b_i}} z_n = \frac{\partial z_n}{\partial \mathbf{z_i}}$$

```python
def ComputeGradients(L, x, z):
    n = len(L)
    z_full = [x] + z

    grad_z = 1
    gradients = []

    for i in range(n, 0, -1):
        W_i, b_i = L[i-1]
        grad_b_i = grad_z
        grad_W_i = grad_z @ z_full[i-1].T
        gradients.append((grad_W_i, grad_b_i))
        grad_z = W_i.T @ grad_z  # propagate backward

    return list(reversed(gradients))
```

General formula for $n \geq 2$:

$$\nabla_{\mathbf{W_i}} z_n = \left(\prod_{j=n}^{i+1} \mathbf{W_j}^T \right)^T \mathbf{z_{i-1}}^T$$

$$\nabla_{\mathbf{b_i}} z_n = \left(\prod_{j=n}^{i+1} \mathbf{W_j}^T \right)^T$$

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/76771/groups#tab-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.

https://canvas.wpi.edu/courses/76771/groups#tab-14853 https://canvas.wpi.edu/courses/76771/groups#tab-14853