Problem 1

Use back-propagation to calculate the gradients of

$$f(W,x) = ||\sigma(Wx)||^2$$

with respect to x and W. Here, $\|\cdot\|^2$ is the calculation of L2 loss, W is a 3×3 matrix, and x is a 3×1 vector, and $\sigma(\cdot)$ is the ReLU function that performs element-wise operation.

We can first write out W and x

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Let's say:

$$z = \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

so to do $a = \sigma(z)$

$$a = \begin{bmatrix} max(0, z_1) \\ max(0, z_2) \\ max(0, z_3) \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{cases} z_i & z_i > 0 \\ 0 & z_i \leq 0 \end{cases}$$

Now we are left with

$$f(W,x) = ||a||^2$$

then Gradient with respect to $\mathbf{a}$

$$\frac{\partial f}{\partial a} = \frac{\partial}{\partial a}(a_1^2 + a_2^2 + a_3^2) = 2a = \begin{bmatrix} 2a_1 \\ 2a_2 \\ 2a_3 \end{bmatrix}$$

so we get the gradient of f with respect to a

$$\nabla_a f = 2a$$

then we want to find $\nabla_z f$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial z}$$

and we know derivative of the ReLU is:

$$\frac{\partial a}{\partial z} = \begin{cases} 1 & z_i > 0 \\ 0 & z_i \leq 0 \end{cases} = \begin{bmatrix} I_{(z_1>0)} & 0 & 0 \\ 0 & I_{(z_2>0)} & 0 \\ 0 & 0 & I_{(z_3>0)} \end{bmatrix}$$

so we can get

$$\nabla_z f = \frac{\partial f}{\partial z} = \begin{bmatrix} 2a_1 \cdot I(z_1 > 0) \\ 2a_2 \cdot I(z_2 > 0) \\ 2a_3 \cdot I(z_3 > 0) \end{bmatrix} = \begin{bmatrix} 2a_1 \; if \; z_1 > 0, \; else \; 0 \\ 2a_2 \; if \; z_2 > 0, \; else \; 0 \\ 2a_3 \; if \; z_3 > 0, \; else \; 0 \end{bmatrix} = 2a \cdot I_{z>0}$$

Now to find $\nabla_x f$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial x}$$

we know can find the $\frac{\partial z}{\partial x}$ which is :

$$\frac{\partial z_k}{\partial x_i} = W_{k,i}$$

so

$$\nabla_x f = \frac{\partial f}{\partial x} = \sum_j \frac{\partial f}{\partial z_j} \frac{\partial z_j}{\partial x_i} = \sum_j 2a \cdot I_{(z_i>0)} \sum_{j=1}^{3} W_{i,j} = \nabla_z f \cdot \sum_{j=1}^{3} W_{i,j}$$

on the other hand $\nabla_W f$ is much easier to find

$$\nabla_W f = \nabla_z f \cdot x^T$$

Problem 2

In this problem, you need to use Gradient Descent (GD) to train the linear classifier in the HW1, i.e., find the parameters W , and then use it to recognize handwritten digits. Adopt still "Cross Entropy" as the loss function.

Requirements:

1. manually derive the gradients of linear classifier when using cross-entropy as the loss function, and write codes to implement it in recognizing handwritten digits
2. the test accuracy should be at least 85%

In [1]:
```
import numpy as np
from urllib import request
import gzip
import pickle
from IPython.display import clear_output
```

In [2]:
```
filename = [
```

```
["training_images","train-images-idx3-ubyte.gz"],
["test_images","t10k-images-idx3-ubyte.gz"],
["training_labels","train-labels-idx1-ubyte.gz"],
["test_labels","t10k-labels-idx1-ubyte.gz"]
]


def download_mnist():
    base_url = "https://ossci-datasets.s3.amazonaws.com/mnist/"
    for name in filename:
        print("Downloading "+name[1]+"...")
        request.urlretrieve(base_url+name[1], name[1])
    print("Download complete.")

def save_mnist():
    mnist = {}
    for name in filename[:2]:
        with gzip.open(name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8, offset=16)
    for name in filename[-2:]:
        with gzip.open(name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8, offset=8)
    with open("mnist.pkl", 'wb') as f:
        pickle.dump(mnist,f)
    print("Save complete.")

def init():
    download_mnist()
    save_mnist()
#    print ((load()[0]).shape)
def load():
    with open("mnist.pkl",'rb') as f:
        mnist = pickle.load(f)
    return mnist["training_images"], mnist["training_labels"], mnist["tes

init()
```

```
Downloading train-images-idx3-ubyte.gz...
Downloading t10k-images-idx3-ubyte.gz...
Downloading train-labels-idx1-ubyte.gz...
Downloading t10k-labels-idx1-ubyte.gz...
Download complete.
Save complete.
```

In [3]:
```
x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000, 28 * 28)/255
y_train = y_train.reshape(60000)
x_test = x_test.reshape(10000, 28 * 28)/255
y_test = y_test.reshape(10000)
```

Linear Classifier Class Definition

---

hand calculed gradients

$$f(W, x, b) = CrossEntropy(SoftMax(Wx + b))$$

need to find $\nabla_W f$ $\nabla_b F$

$z = Wx + b$

$s = softmax(z)$

$\mathcal{L} = crossentropy(s, y)$

- where y is the ground truth

k = number of output classes

$$\frac{\partial s_i}{\partial z_k} = \frac{\partial}{\partial z_k}(\frac{e^{s_i}}{\sum_j e^{s_j}}) = s_i(\delta_{ik} - s_k)$$

$$\frac{\partial \mathcal{L}}{\partial s_i} = \frac{\partial}{\partial s_i}(-\sum_i y_i \log(s_i)) = -\frac{y_i}{s_i}$$

$$\frac{\partial \mathcal{L}}{\partial z_k} = \frac{\partial \mathcal{L}}{\partial s_i}\frac{\partial s_i}{\partial z_k} = -\frac{y_i}{s_i} s_i(\delta_{ik} - s_k) = s_k - y_k$$

$$\frac{\partial \mathcal{L}}{\partial z} = s - y$$

$$\frac{\partial z}{\partial W} = x^T$$

$$\frac{\partial z}{\partial b} = \delta = I$$

$$\frac{\partial \mathcal{L}}{\partial W} = (s - y)x^T$$

$$\frac{\partial \mathcal{L}}{\partial b} = (s - y)$$

In [4]:
```
class LinearClassifier:
    def __init__(self, inputDim=784, outputDim=10, lr=0.01):
        # Initialize weights and bias
        self.W = np.random.randn(inputDim, outputDim) * 0.01  # small ran
        self.b = np.zeros(outputDim)
        self.lr = lr

    def forward(self, x):
        # Linear forward pass
        scores = x @ self.W + self.b
        return scores

    def softmax(self, s):
        exps = np.exp(s - np.max(s, axis=1, keepdims=True))
        probs = exps / np.sum(exps, axis=1, keepdims=True)
        return probs

    def predict(self, data):
        scores = self.forward(data)
        probs = self.softmax(scores)
        predictions = np.argmax(probs, axis=1)
        return predictions, probs

    def crossEntropyLoss(self, probs, labels):
```

```
        N=len(labels)
        loss = -np.mean(np.log(probs[np.arange(N), labels] + 1e-8))
        return loss


    def backward(self, x, y, probs):
        N = len(y)
        # Compute gradient for softmax with cross-entropy loss
        dscores = probs.copy()
        dscores[np.arange(N), y] -= 1  # Subtract 1 from correct class

        # Compute gradients with respect to W and b
        dW = x.T @ dscores
        db = np.sum(dscores, axis=0)

        # Update parameters using gradient descent
        self.W -= self.lr * dW
        self.b -= self.lr * db

        return dW, db

    def sgd(self, x, y, epochs=10, batch_size=128):
        N = len(y)
        for epoch in range(epochs):
            epoch_loss = 0
            for i in range(0, N, batch_size):
                x_batch = x[i:i+batch_size]
                y_batch = y[i:i+batch_size]
                scores = self.forward(x_batch)
                probs = self.softmax(scores)
                loss = self.crossEntropyLoss(probs, y_batch)
                self.backward(x_batch, y_batch, probs)
            clear_output(wait=True)
            print(f"Epoch {epoch+1:2.0f}, Loss: {loss:.3f}")

        return self.W, self.b


    def accuracy(self, X, y):
        predictions, _ = self.predict(X)
        return np.mean(predictions == y)
```

In [5]:

```
model = LinearClassifier()
w, b = model.sgd(x_train, y_train, batch_size=128)

train_accuracy = model.accuracy(x_train, y_train)
print(f"Train Accuracy: {train_accuracy*100:3.2f}%")

test_accuracy = model.accuracy(x_test, y_test)
print(f"Test Accuracy : {test_accuracy*100:3.2f}%")
```

```
Epoch 10, Loss: 0.268
Train Accuracy: 90.03%
Test Accuracy : 89.35%
```