Homework 3: Differentiable Programming

```
In [ ]: import torch
        import matplotlib.pyplot as plt
```

1. Edge Cases of Automatic Differentiation

We will construct some cases where PyTorch returns derivatives that make no sense. The underlying problem is that PyTorch does not sanitize its inputs; it relies on the users to make sure the inputs to automatic differentiation are well-defined mathematically. You might find it helpful to go over this week's demo again to revisit the difference between a mathematical function and a DiffProg function.

NOTE: For each exercise in the homework, write a vanilla Python function and compute its derivative as returned by PyTorch's automatic differentiation engine. Do not write your own `torch.autograd.Function` implementation (that would defeat the purpose of the homework).

1.1 Recall that if a (mathematical) function $f : \mathbb{R} \to \mathbb{R}$ is discontinuous at a point $\hat{x}$, then it cannot be differentiable at $\hat{x}$.

- Define and plot a (mathematical) function $f : \mathbb{R} \to \mathbb{R}$ which is discontinuous at $\hat{x}$ with a jump discontinuity. Clearly show the point at which f is discontinuous and indicate whether it is right continuous or left continuous. Look at https://upload.wikimedia.org/wikipedia/commons/6/68/Detachment_example.gif for an example of a jump discontinuity.
- Implement f as a DiffProg function in PyTorch so that PyTorch returns a derivative of 0 at $\hat{x}$, our point of discontinuity.
- Implement f again in DiffProg so that PyTorch now returns a derivative of -1728 at exactly the same point $\hat{x}$.

Note that the derivative of f is not even defined at $\hat{x}$. Yet, we can get it to return two different values of the derivative.

Hint: Use if statements. Implement the first DiffProg function with two branches one for $x \leq \hat{x}$ and the other $x > \hat{x}$. Implement the second DiffProg function using three branches $x < \hat{x}$, $x > \hat{x}$ and $x = \hat{x}$ and try to change the third branch to obtain the desired outcome.

Solution:

a. We define a function as follows for a function with derivative 0 at $\hat{x} = 0$:

$$f(x) = \begin{cases} x^3; x \in (-\infty, 0] \\ x^3 + 200; x \in (0, \infty) \end{cases}$$

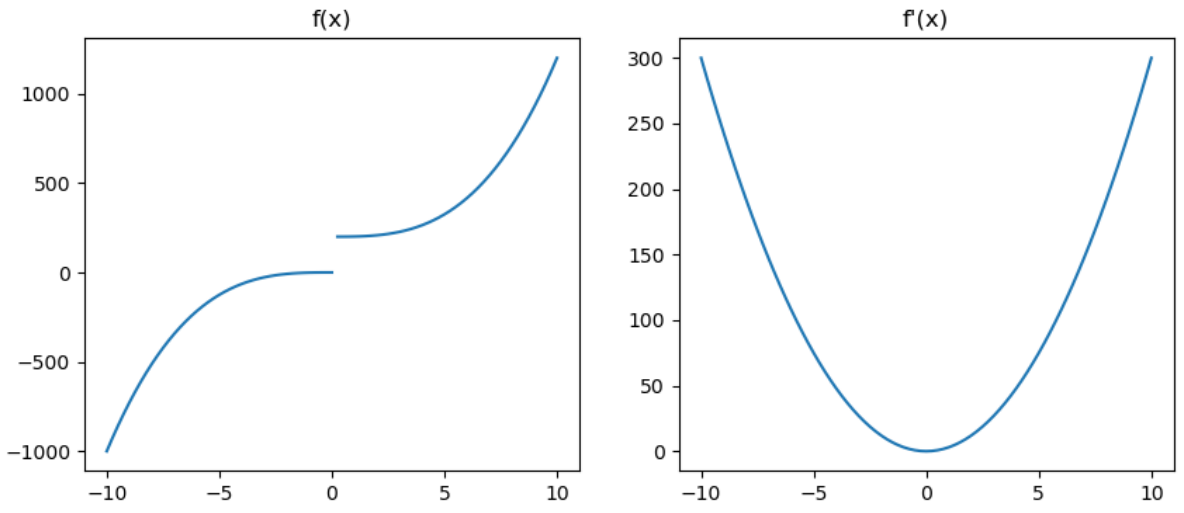
This function is discontinuous at $x = 0$ and we have defined it as left continuous.

b. We define a function as follows for a function with derivative -1728 at $\hat{x} = 0$:

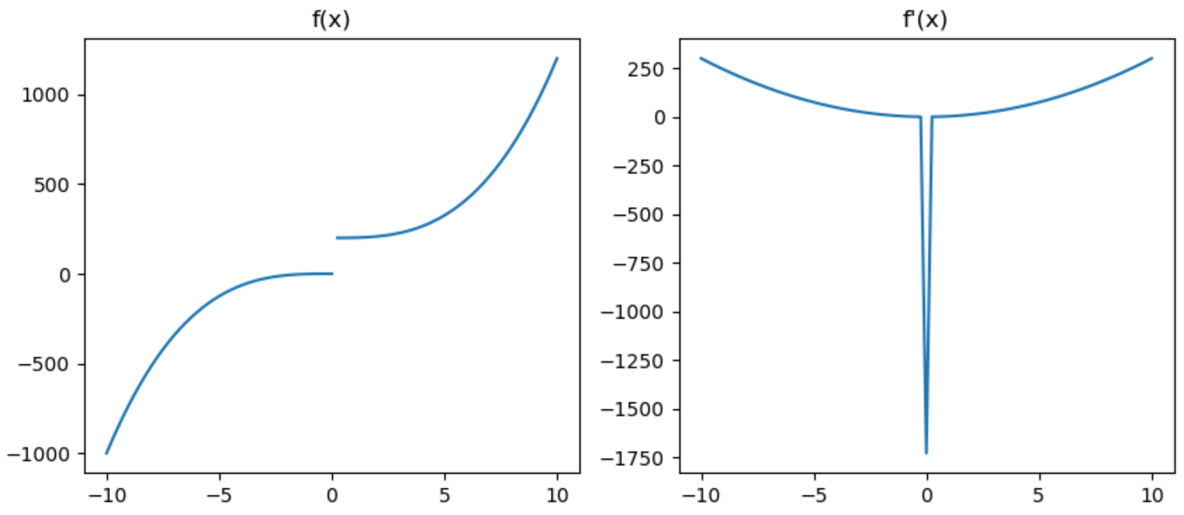
$$f(x) = \begin{cases} x^3; x \in (-\infty, 0) \\ x^3 + 200; x \in (0, \infty) \\ -1728 * x; x = 0 \end{cases}$$

In [ ]:
```
def my_jump_function(x):
    if x <= 0:
        return x**3
    else:
        return x**3+200

def my_jump_function_2(x):
    if x < 0:
        return x**3
    elif x > 0:
        return x**3+200
    else:
        return -1728*x

def compute_f_and_df(func, jump=True, xs = torch.linspace(-10, 10, 81, requi

    function_values = []
    derivatives = []
    for x in xs:
        y = func(x, **kwargs)
        function_values.append(y.item())
        y_prime = torch.autograd.grad(outputs = y, inputs = [x], allow_unuse
        if y_prime is None:
            y_prime = 0.0
        derivatives.append(y_prime)

    f, ax = plt.subplots(1, 2, figsize=(10, 4))
    if jump is True:
        ax[0].plot(xs.detach().numpy()[:41], function_values[:41])
        ax[0].plot(xs.detach().numpy()[41:], function_values[41:], color='ta
    else:
        ax[0].plot(xs.detach().numpy(), function_values)
    ax[0].set_title("f(x)")
    ax[1].plot(xs.detach().numpy(), derivatives)
    ax[1].set_title("f'(x)")
```

In [ ]:
```
compute_f_and_df(my_jump_function)
```



In [ ]:
```
# Visualizing function where we define -1728 as derivative for x=0
compute_f_and_df(my_jump_function_2)
```



1.2 Inconsistent derivatives of a differentiable function.

Consider the (mathematical) function $g(x) = x^4$. Clearly, g is differentiable everywhere.

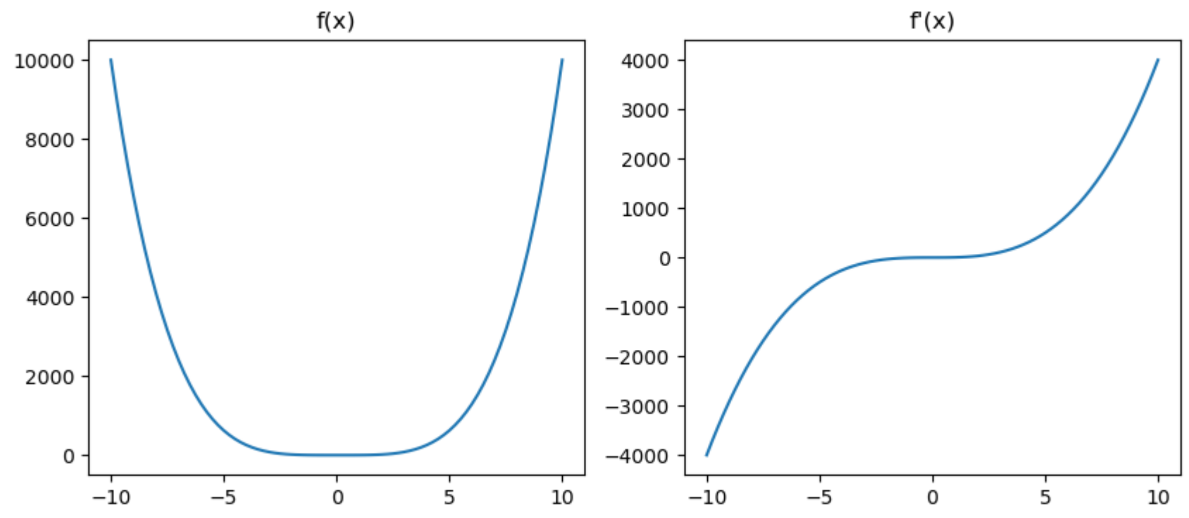
- Implement g as a DiffProg function in PyTorch so that PyTorch returns a derivative of 0 at $\hat{x} = 0$.
- Implement g again in DiffProg so that PyTorch now returns a derivative of 897 at exactly the same point $\hat{x} = 0$.

The takeaway message of this exercise is that the data scientist must make sure the inputs to automatic differentiation are well-defined mathematically.

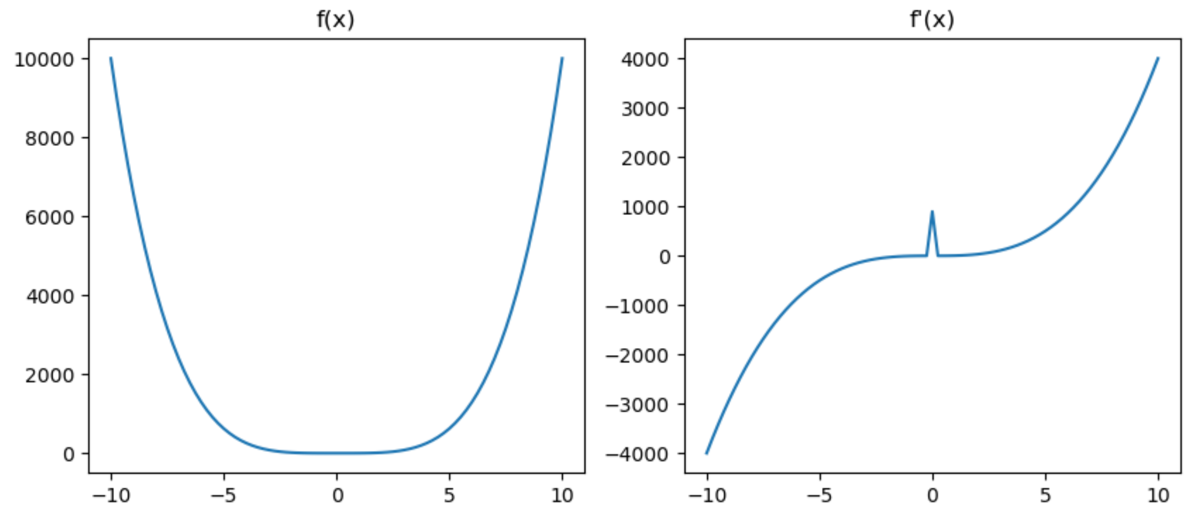
Hint: Use branches again. For the second function, use two branches $x = 0$ and $x \neq 0$.

In [ ]:
```
# Should return derivative of 0 at x=0
def g_1(x):
    return x**4
```

```
# Should return derivative of 897 at x=0
def g_2(x):
    if x==0:
        return 897*x
    else:
        return x**4

compute_f_and_df(g_1, jump=False)
```



In [ ]:
```
compute_f_and_df(g_2, jump=False)
```



1.3 Derivatives with loops: When is it valid?

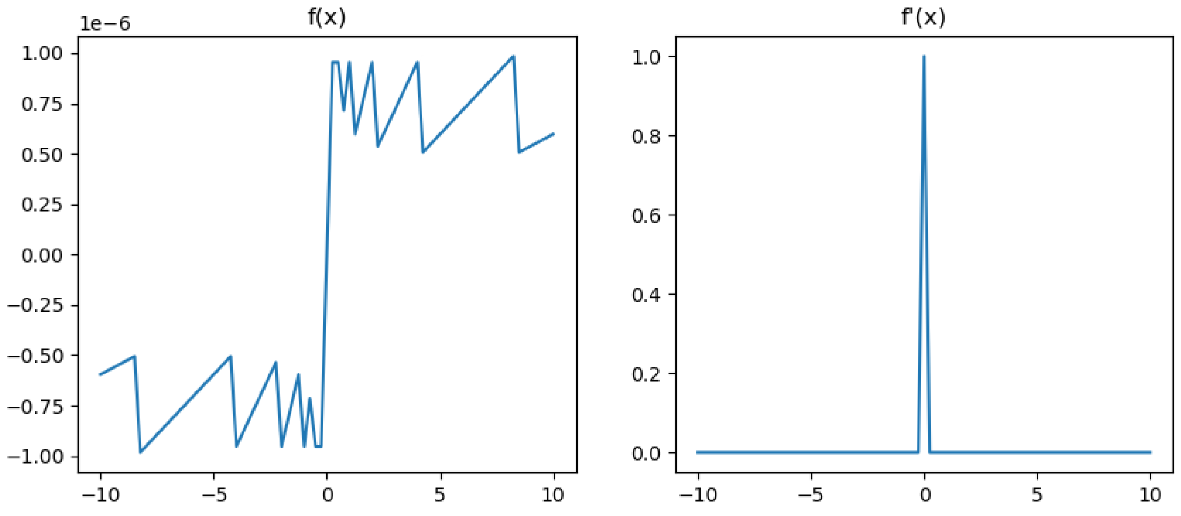
In the lab, we defined a (mathematical) function $h(x,n) = \sum_{i=1}^{n} x^{n-1}$. We implemented this in DiffProg using a loop such that automatic differentiation gives us $\partial h(x,n)$ correctly. In this exercise, we will define ∂x a DiffProg function with a loop so that the underlying mathematical function is discontinuous.

- Write a DiffProg function in PyTorch which takes an input x_0 and iteratively updates $x_{t+1} \leftarrow x_t/2$ until a stopping criterion $|x_t| < 10^{-6}$ is satisfied.

- Plot this function in the range $[-2,2]$. Are the derivatives of this function well-defined everywhere?
- Find a point $\hat{x}$ such that implementing the stopping criterion as $|x_t| < 10^{-6}$ or $|x_t| \leq 10^{-6}$ changes the value of the derivative returned by PyTorch. Is the derivative mathematically well-defined at $\hat{x}$?
- Write out the (mathematical) function $\psi : \mathbb{R} \to \mathbb{R}$ which is implemented by this DiffProg function.

The takeaway message of this part is that one must be careful when defining DiffProg functions with loops. The stopping criterion of the loop must not depend on the input which respect to which we compute a derivative.

In [ ]:
```
# Writing DiffProg function and plotting it for [-2,2]

def my_func_discontinuous(x0, condition='<'):
    if condition == '<':
        while torch.abs(x0) >= 1e-6:
            x0 = x0/2
    elif condition == '<=':
        while torch.abs(x0) > 1e-6:
            x0 = x0/2
    else:
        raise ValueError #"Must be < or <="
    return x0

compute_f_and_df(my_func_discontinuous, jump=False)
```



We see that in the range $[-2,2]$ the derivatives are not well defined as we see a lot of discontinuities at various points in the plot such as $x = \pm 1$. There is also a jump discontinuity at x=0.

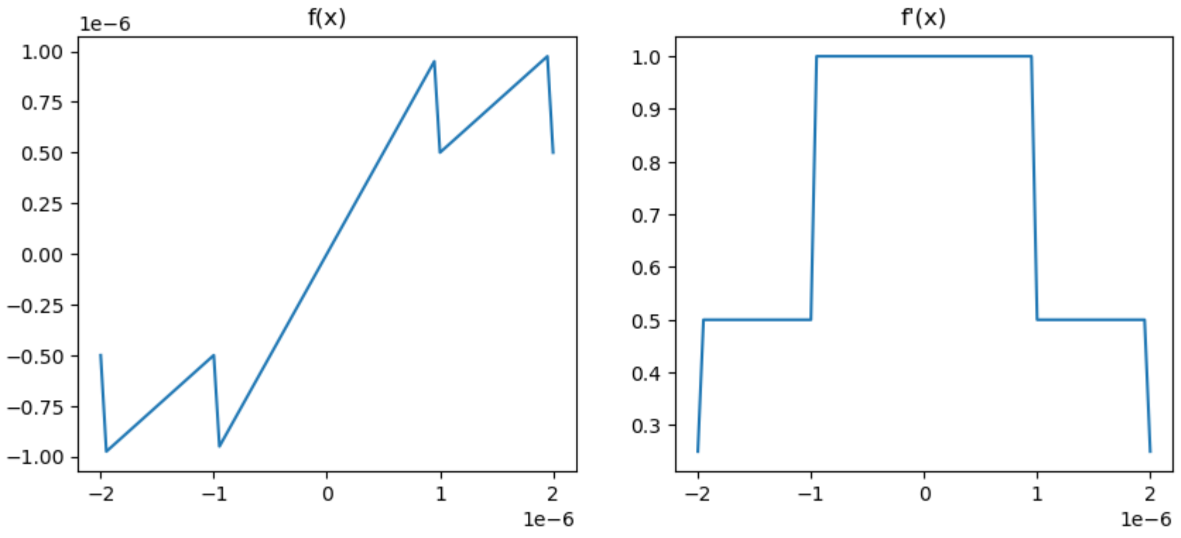
In [ ]:
```
# Since we are changing the stopping condition, the derivative at the stoppi
x = torch.tensor(1e-6, requires_grad=True)
y = my_func_discontinuous(x,condition='<')
grad = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
```

```
print(x, grad, "(using < in terminating condition)")

y = my_func_discontinuous(x,condition='<=')
grad = torch.autograd.grad(outputs=y, inputs=[x], allow_unused=True)[0]
print(x, grad, "(using <= in terminating condition)")
```

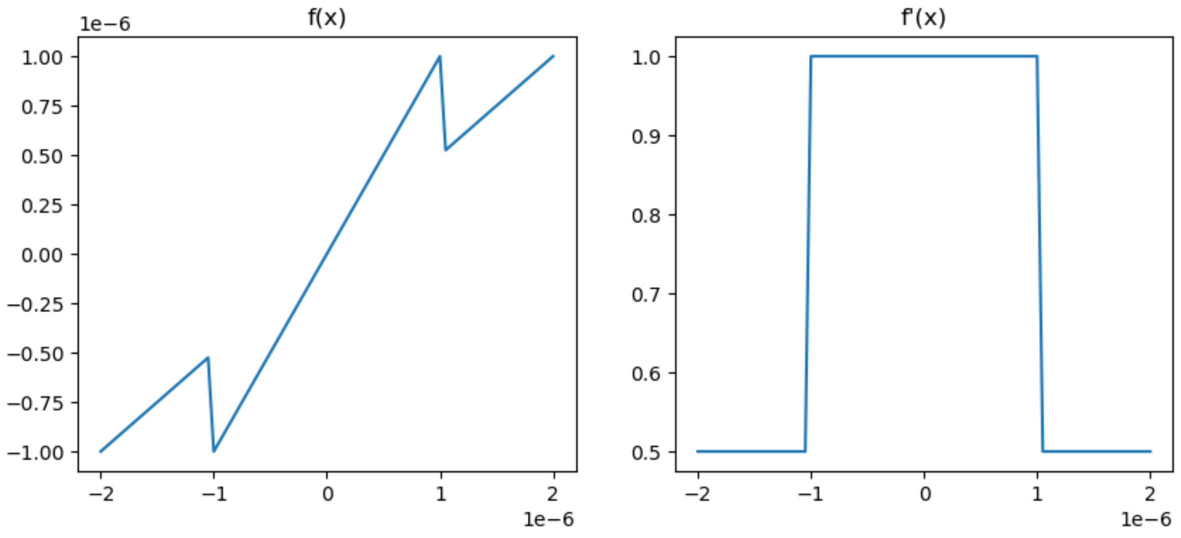
```
tensor(1.0000e-06, requires_grad=True) tensor(0.5000) (using < in terminati
ng condition)
tensor(1.0000e-06, requires_grad=True) tensor(1.) (using <= in terminating
condition)
```

We see that our assumption is correct. We now plot the derivative at this point for both the conditions to see whether the derivative is well defined.

```
In [ ]: compute_f_and_df(my_func_discontinuous, jump=False, xs=torch.linspace(-2e-6,
```



```
In [ ]: compute_f_and_df(my_func_discontinuous, jump=False, xs=torch.linspace(-2e-6,
```



We see that the derivative is not well defined at |1e-6| as there is a jump between 0.5 and 1

The mathematical function implemented by this DiffProg function is as follows:

$$f(x) = \begin{cases} f(x/2); \text{ if } x \geq 10^{-6} \\ 0; \text{otherwise} \end{cases}$$

1.4 When can we not use branches in differentiable programs?

Consider the mathematical function $\phi : \mathbb{R} \to [0, 1]$ by $$

\phi(x) = \begin{cases} 1 \text{ if } x \ge 0 \ 0 \text{ otherwise}

\end{cases} $$

- Plot this function. Is it continuous? Is it differentiable almost everywhere?
- Implement this in PyTorch. Try to compute its derivatives. What do we get?
- Can we train a differentiable program containing this function as a component using stochastic gradient descent? Why or why not? Justify your answer in words.

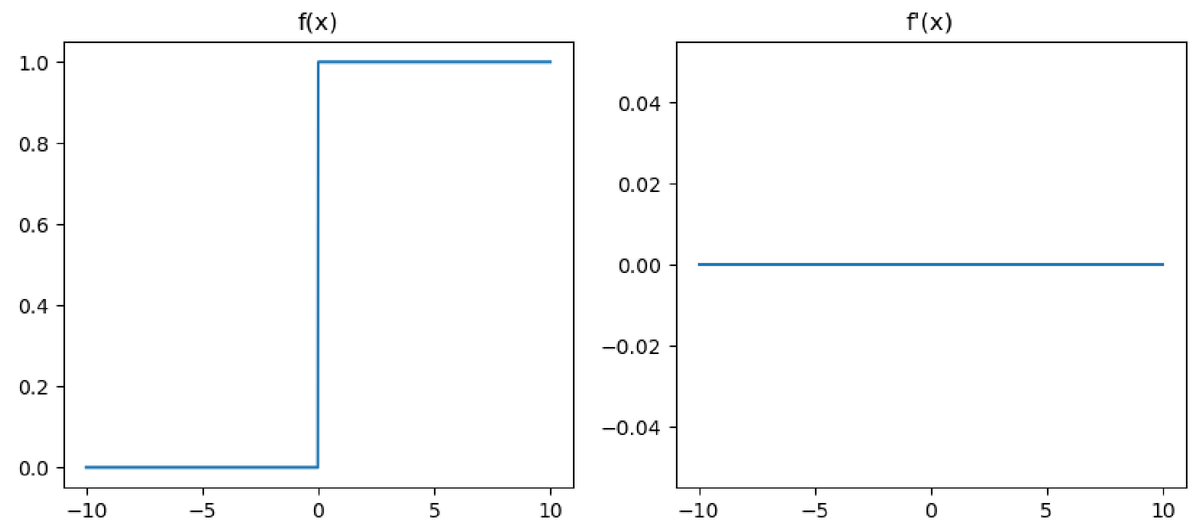
Note: The classification accuracy of a binary classifier can be computed using the function ϕ. Why d o we use logistic regression to train a classifier and not use the classification accuracy directly?

Solution:

This function is continuous and differentiable almost everywhere, with the exception of $x = 0$ where we observe a jump discontinuity.

We plot it along with its derivative after implementing it as a DiffProg in PyTorch (in the interest of resuing code!)

In [ ]:
```
def my_func_0_1(x):
    if x >= 0:
        return torch.ones_like(x, requires_grad=True)
    else:
        return torch.zeros_like(x, requires_grad=True)

compute_f_and_df(my_func_0_1, jump=False, xs=torch.linspace(-10,10,1001,requ
```



We get a derivative of 0 over all ranges of values.

We cannot train a stochastic gradient descent model using this function as a component because this function has a gradient of 0. Due to this we will always have an update of value of 0 and stoachstic gradient descent will be unable to update the parameters.

2. Data Augmentation

Data augmentation can be applied at training time or testing time.

- Training time: in each iteration, we sample a minibatch, take one transformation per-image and use those instead to compute the minibatch stochastic gradient. The rest of the training loop continues as usual.
- Test time: we predict an output for an image x as follows. Take augmentations $x_1, x_2, \cdots, x_T$ of x. For each augmented image x_i, obtain prediction y_i. The combined prediction y for image x is obtained by taking a majority vote from $y_1, \cdots, y_T$. Note that the augmentations can only be used to compute the accuracy but not the loss.

In this exercise, we will try four combinations:

1. No data augmentation for training or testing
2. Use data augmentation for training but not for testing
3. Use data augmentation for testing but not for training
4. Use data augmentation for both training and testing

Here are the details:

- The setup is identical to the lab. Take the FashionMNIST dataset and randomly subsample 12% of its training set to work with. As a test set, we will use the full test set of FashionMNIST.
- We will use a convolutional neural network defined in the lab.

- Use a batch size of 16 and a learning rate of 0.04.
- Train the model for 100 passes through the data or until you observe perfect interpolation of the training data (i.e., the training accuracy is 100%).
- We will use a random crop and a random rotation as our transformations.
- For testing time, use T = 8 augmentations for each image.

The deliverables are:

1. Report the final test accuracy for each of the 4 settings considered above.
2. Make 4 plots, one each for the train loss, train accuracy, test loss and test accuracy over the course of training (i.e., the metric on the y-axis and number of effective passes on the x-axis). Plot all 4 lines on the same plot.

Hint: You may use the function "transform_selected_data" defined in this week's demo to perform the data augmentations.

In [ ]:
```
import numpy as np
import pandas as pd
from torchvision.datasets import FashionMNIST
from torch.nn.functional import cross_entropy
import time

# Fix the random seeds for reproducibility
torch.manual_seed(0)
np.random.seed(1)
```

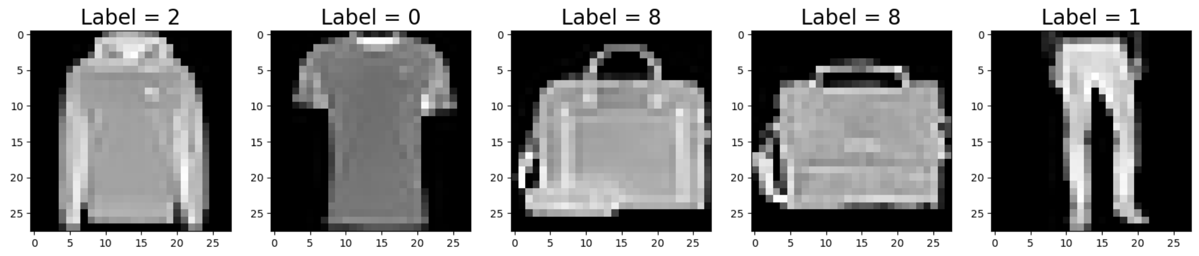
2.1 Loading the MNIST dataset and setting up the CNN

We load the MNIST dataset and set up some helper functions to preprocess the data and train the model

In [ ]:
```
from torchvision.datasets import FashionMNIST
import numpy as np
import matplotlib.pyplot as plt
import torchvision.transforms as transforms

# download dataset (~117M in size)
train_dataset = FashionMNIST('../../data', train=True, download=False)
X_train = train_dataset.data # torch tensor of type uint8
y_train = train_dataset.targets # torch tensor of type Long
test_dataset = FashionMNIST('../../data', train=False, download=False)
X_test = test_dataset.data
y_test = test_dataset.targets

# choose a subsample of 10% of the data:
idxs_train = torch.from_numpy(
    np.random.choice(X_train.shape[0], replace=False, size=int(X_train.shape
X_train, y_train = X_train[idxs_train], y_train[idxs_train]
# idxs_test = torch.from_numpy(
#     np.random.choice(X_test.shape[0], replace=False, size=X_test.shape[0]/
# X_test, y_test = X_test[idxs_test], y_test[idxs_test]
```

```
print(f'X_train.shape = {X_train.shape}')
print(f'n_train: {X_train.shape[0]}, n_test: {X_test.shape[0]}')
print(f'Image size: {X_train.shape[1:]}')

f, ax = plt.subplots(1, 5, figsize=(20, 4))
for i, idx in enumerate(np.random.choice(X_train.shape[0], 5)):
    ax[i].imshow(X_train[idx], cmap='gray', vmin=0, vmax=255)
    ax[i].set_title(f'Label = {y_train[idx]}', fontsize=20)

# Normalize dataset: pixel values lie between 0 and 255
# Normalize them so the pixelwise mean is zero and standard deviation is 1

X_train = X_train.float()  # convert to float32
X_train = X_train.view(-1, 784)
mean, std = X_train.mean(axis=0), X_train.std(axis=0)
X_train = (X_train - mean[None, :]) / (std[None, :] + 1e-6)  # avoid divide

X_test = X_test.float()
X_test = X_test.view(-1, 784)
X_test = (X_test - mean[None, :]) / (std[None, :] + 1e-6)

n_class = np.unique(y_train).shape[0]
```

```
X_train.shape = torch.Size([7200, 28, 28])
n_train: 7200, n_test: 10000
Image size: torch.Size([28, 28])
```



In [ ]:
```
# Use this for your homework

def transform_selected_data(X):
    # X is of shape (B, 784), where B is the batch_size
    X = X.view(-1, 28, 28)  # reshape to 28x28
    transform1 = transforms.RandomResizedCrop((28, 28), scale=(0.75, 1.0), r
    transform2 = transforms.RandomRotation((-10, 10))
    X_transformed = transform2(transform1(X))
    return X_transformed.view(-1, 784) # reshape into a vector


# call, e.g., as `transform_selected_data(X_train[:10])`
```

In [ ]:
```
from torch.nn.functional import cross_entropy

def compute_objective(net, X, y):
    """ Compute the multinomial logistic loss.
        net is a module
        X of shape (n, d) and y of shape (n,)
    """
    # send
    score = net(X)
    # PyTorch's function cross_entropy computes the multinomial logistic los
    return cross_entropy(input=score, target=y, reduction='mean')

@torch.no_grad()
def compute_accuracy(net, X, y):
    """ Compute the classification accuracy
        ws is a list of tensors of consistent shapes
        X of shape (n, d) and y of shape (n,)
    """
    score = net(X)
    predictions = torch.argmax(score, axis=1)  # class with highest score is
    # Return the fraction of predictions that are correct
    return (predictions == y).sum() * 1.0 / y.shape[0]

@torch.no_grad()
def compute_logs(net, verbose=False):
    train_loss = compute_objective(net, X_train, y_train)
    test_loss = compute_objective(net, X_test, y_test)
    train_accuracy = compute_accuracy(net, X_train, y_train)
    test_accuracy = compute_accuracy(net, X_test, y_test)
    if verbose:
        print(('Train Loss = {:.3f}, Train Accuracy = {:.3f}, ' +
               'Test Loss = {:.3f}, Test Accuracy = {:.3f}').format(
                train_loss.item(), train_accuracy.item(),
                test_loss.item(), test_accuracy.item())
    )
    return (train_loss, train_accuracy, test_loss, test_accuracy)


def minibatch_sgd_one_pass(net, X, y, learning_rate, batch_size, verbose=Fal
    num_examples = X.shape[0]
    average_loss = 0.0
    num_updates = int(round(num_examples / batch_size))
    for i in range(num_updates):
        # TODO: your code here: sample `batch_size` many indices from {0, ..
        idxs = np.random.choice(num_examples, size=(batch_size,))
        # compute the objective.
        objective = compute_objective(net, X[idxs], y[idxs])
        average_loss = 0.99 * average_loss + 0.01 * objective.item()
        if verbose and (i+1) % 100 == 0:
            print(average_loss)

        # TODO: your code here: compute the gradient using automatic differe
        # Hint: you can access the parameters of `net.parameters()`
        gradients = torch.autograd.grad(outputs=objective, inputs=net.parame

        # perform SGD update. IMPORTANT: Make the update inplace!
        # Hint: you can access the parameters of `net.parameters()`
        with torch.no_grad():
            for (w, g) in zip(net.parameters(), gradients):
                w -= learning_rate * g
    return net
```

We will use a ConvNet written as a PyTorch module.

```
In [ ]: class MyConvNet(torch.nn.Module):
            def __init__(self, num_classes=10):
                super().__init__()
                self.conv_ensemble_1 = torch.nn.Sequential(
                    torch.nn.Conv2d(1, 16, kernel_size=5, padding=2),
                    torch.nn.ReLU(),
                    torch.nn.MaxPool2d(2))
                self.conv_ensemble_2 = torch.nn.Sequential(
                    torch.nn.Conv2d(16, 32, kernel_size=5, padding=2),
                    torch.nn.ReLU(),
                    torch.nn.MaxPool2d(2))
                self.fully_connected_layer = torch.nn.Linear(7*7*32, 10)
                # Note: the size 7*7*32 comes from the output size of the
                # previous layer. We will see how to find this next week.

            def forward(self, x):
                x = x.view(-1, 1, 28, 28)  # reshape input; convolutions need a char
                out = self.conv_ensemble_1(x)  # first convolution + relu + pooling
                out = self.conv_ensemble_2(out) # second convolution + relu + poolir
                out = out.view(out.shape[0], -1)  # flatten output
                out = self.fully_connected_layer(out)  # output layer
                return out
```

2.2 Training and Test without data augmentations

```
In [ ]: learning_rate = 0.04

        logs = []

        model = MyConvNet(num_classes=10)
        print('Iteration 0', end=', ')
        logs.append(compute_logs(model, verbose=True))

        batch_size = 16

        for j in range(100):
            model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
            print(f'Iteration {j+1}', end=', ')
            log = compute_logs(model, verbose=True)
            logs.append(log)
            if log[1] == 1.0:
                break
```

```
Iteration 0, Train Loss = 2.278, Train Accuracy = 0.145, Test Loss = 2.282,
Test Accuracy = 0.144
Iteration 1, Train Loss = 0.507, Train Accuracy = 0.817, Test Loss = 0.544,
Test Accuracy = 0.804
Iteration 2, Train Loss = 0.417, Train Accuracy = 0.847, Test Loss = 0.486,
Test Accuracy = 0.831
Iteration 3, Train Loss = 0.325, Train Accuracy = 0.883, Test Loss = 0.436,
Test Accuracy = 0.854
Iteration 4, Train Loss = 0.316, Train Accuracy = 0.885, Test Loss = 0.463,
Test Accuracy = 0.842
Iteration 5, Train Loss = 0.271, Train Accuracy = 0.901, Test Loss = 0.410,
Test Accuracy = 0.864
Iteration 6, Train Loss = 0.246, Train Accuracy = 0.914, Test Loss = 0.446,
Test Accuracy = 0.865
Iteration 7, Train Loss = 0.262, Train Accuracy = 0.903, Test Loss = 0.498,
Test Accuracy = 0.846
Iteration 8, Train Loss = 0.194, Train Accuracy = 0.933, Test Loss = 0.446,
Test Accuracy = 0.863
Iteration 9, Train Loss = 0.158, Train Accuracy = 0.946, Test Loss = 0.423,
Test Accuracy = 0.878
Iteration 10, Train Loss = 0.171, Train Accuracy = 0.941, Test Loss = 0.49
0, Test Accuracy = 0.864
Iteration 11, Train Loss = 0.135, Train Accuracy = 0.950, Test Loss = 0.47
4, Test Accuracy = 0.868
Iteration 12, Train Loss = 0.103, Train Accuracy = 0.969, Test Loss = 0.46
6, Test Accuracy = 0.876
Iteration 13, Train Loss = 0.098, Train Accuracy = 0.967, Test Loss = 0.51
6, Test Accuracy = 0.878
Iteration 14, Train Loss = 0.085, Train Accuracy = 0.975, Test Loss = 0.51
0, Test Accuracy = 0.870
Iteration 15, Train Loss = 0.115, Train Accuracy = 0.960, Test Loss = 0.56
9, Test Accuracy = 0.866
Iteration 16, Train Loss = 0.139, Train Accuracy = 0.948, Test Loss = 0.61
9, Test Accuracy = 0.850
Iteration 17, Train Loss = 0.063, Train Accuracy = 0.978, Test Loss = 0.56
2, Test Accuracy = 0.872
Iteration 18, Train Loss = 0.064, Train Accuracy = 0.979, Test Loss = 0.61
2, Test Accuracy = 0.868
Iteration 19, Train Loss = 0.062, Train Accuracy = 0.981, Test Loss = 0.65
7, Test Accuracy = 0.863
Iteration 20, Train Loss = 0.054, Train Accuracy = 0.986, Test Loss = 0.65
3, Test Accuracy = 0.874
Iteration 21, Train Loss = 0.049, Train Accuracy = 0.983, Test Loss = 0.67
2, Test Accuracy = 0.871
Iteration 22, Train Loss = 0.050, Train Accuracy = 0.988, Test Loss = 0.68
9, Test Accuracy = 0.872
Iteration 23, Train Loss = 0.030, Train Accuracy = 0.991, Test Loss = 0.77
9, Test Accuracy = 0.874
Iteration 24, Train Loss = 0.034, Train Accuracy = 0.989, Test Loss = 0.74
9, Test Accuracy = 0.870
Iteration 25, Train Loss = 0.013, Train Accuracy = 0.997, Test Loss = 0.74
1, Test Accuracy = 0.878
Iteration 26, Train Loss = 0.013, Train Accuracy = 0.997, Test Loss = 0.77
1, Test Accuracy = 0.875
Iteration 27, Train Loss = 0.009, Train Accuracy = 0.999, Test Loss = 0.76
9, Test Accuracy = 0.876
```

```
Iteration 28, Train Loss = 0.006, Train Accuracy = 0.999, Test Loss = 0.78
8, Test Accuracy = 0.877
Iteration 29, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 0.82
4, Test Accuracy = 0.876
Iteration 30, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 0.81
4, Test Accuracy = 0.877
Iteration 31, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.83
4, Test Accuracy = 0.878
Iteration 32, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 0.86
6, Test Accuracy = 0.878
Iteration 33, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.86
5, Test Accuracy = 0.878
Iteration 34, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.87
0, Test Accuracy = 0.877
Iteration 35, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.88
3, Test Accuracy = 0.877
```

2.3 Training with Data Augmentations

In [ ]:

```
learning_rate = 0.04

logs_2 = []

model = MyConvNet(num_classes=10)
print('Iteration 0', end=', ')
logs_2.append(compute_logs(model, verbose=True))

batch_size = 16

for j in range(100):
    X_train_augmented = transform_selected_data(X_train)
    model = minibatch_sgd_one_pass(model, X_train_augmented, y_train, learni
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_2.append(log)
    if log[1] == 1.0:
        break
```

```
Iteration 0, Train Loss = 2.316, Train Accuracy = 0.076, Test Loss = 2.316,
Test Accuracy = 0.084
```

```
/Users/hridaybaghar/opt/miniconda3/envs/data598/lib/python3.8/site-package
s/torchvision/transforms/transforms.py:852: UserWarning: Argument interpola
tion should be of type InterpolationMode instead of int. Please, use Interp
olationMode enum.
  warnings.warn(
Iteration 1, Train Loss = 0.708, Train Accuracy = 0.743, Test Loss = 0.741,
Test Accuracy = 0.735
Iteration 2, Train Loss = 1.030, Train Accuracy = 0.658, Test Loss = 1.073,
Test Accuracy = 0.661
Iteration 3, Train Loss = 1.189, Train Accuracy = 0.614, Test Loss = 1.236,
Test Accuracy = 0.599
Iteration 4, Train Loss = 0.676, Train Accuracy = 0.776, Test Loss = 0.764,
Test Accuracy = 0.753
Iteration 5, Train Loss = 0.489, Train Accuracy = 0.833, Test Loss = 0.593,
Test Accuracy = 0.809
Iteration 6, Train Loss = 1.021, Train Accuracy = 0.735, Test Loss = 1.117,
Test Accuracy = 0.722
Iteration 7, Train Loss = 0.436, Train Accuracy = 0.836, Test Loss = 0.534,
Test Accuracy = 0.816
Iteration 8, Train Loss = 0.477, Train Accuracy = 0.812, Test Loss = 0.576,
Test Accuracy = 0.792
Iteration 9, Train Loss = 0.686, Train Accuracy = 0.762, Test Loss = 0.787,
Test Accuracy = 0.747
Iteration 10, Train Loss = 0.364, Train Accuracy = 0.869, Test Loss = 0.48
8, Test Accuracy = 0.842
Iteration 11, Train Loss = 0.401, Train Accuracy = 0.853, Test Loss = 0.53
5, Test Accuracy = 0.826
Iteration 12, Train Loss = 0.443, Train Accuracy = 0.846, Test Loss = 0.54
9, Test Accuracy = 0.819
Iteration 13, Train Loss = 0.362, Train Accuracy = 0.869, Test Loss = 0.48
5, Test Accuracy = 0.839
Iteration 14, Train Loss = 0.459, Train Accuracy = 0.837, Test Loss = 0.61
9, Test Accuracy = 0.802
Iteration 15, Train Loss = 0.558, Train Accuracy = 0.804, Test Loss = 0.72
4, Test Accuracy = 0.775
Iteration 16, Train Loss = 0.406, Train Accuracy = 0.863, Test Loss = 0.55
4, Test Accuracy = 0.831
Iteration 17, Train Loss = 0.369, Train Accuracy = 0.870, Test Loss = 0.52
1, Test Accuracy = 0.826
Iteration 18, Train Loss = 0.392, Train Accuracy = 0.856, Test Loss = 0.57
7, Test Accuracy = 0.826
Iteration 19, Train Loss = 0.302, Train Accuracy = 0.891, Test Loss = 0.50
4, Test Accuracy = 0.847
Iteration 20, Train Loss = 0.276, Train Accuracy = 0.901, Test Loss = 0.44
8, Test Accuracy = 0.859
Iteration 21, Train Loss = 0.589, Train Accuracy = 0.793, Test Loss = 0.74
8, Test Accuracy = 0.772
Iteration 22, Train Loss = 0.449, Train Accuracy = 0.848, Test Loss = 0.65
1, Test Accuracy = 0.812
Iteration 23, Train Loss = 0.361, Train Accuracy = 0.864, Test Loss = 0.49
3, Test Accuracy = 0.835
Iteration 24, Train Loss = 0.383, Train Accuracy = 0.863, Test Loss = 0.55
9, Test Accuracy = 0.824
Iteration 25, Train Loss = 0.362, Train Accuracy = 0.866, Test Loss = 0.55
3, Test Accuracy = 0.818
Iteration 26, Train Loss = 0.488, Train Accuracy = 0.844, Test Loss = 0.70
6, Test Accuracy = 0.803
Iteration 27, Train Loss = 0.477, Train Accuracy = 0.826, Test Loss = 0.68
9, Test Accuracy = 0.782
Iteration 28, Train Loss = 0.382, Train Accuracy = 0.861, Test Loss = 0.60
2, Test Accuracy = 0.815
```

```
Iteration 29, Train Loss = 0.249, Train Accuracy = 0.911, Test Loss = 0.46
0, Test Accuracy = 0.863
Iteration 30, Train Loss = 0.334, Train Accuracy = 0.882, Test Loss = 0.53
5, Test Accuracy = 0.844
Iteration 31, Train Loss = 0.380, Train Accuracy = 0.872, Test Loss = 0.63
1, Test Accuracy = 0.822
Iteration 32, Train Loss = 0.408, Train Accuracy = 0.854, Test Loss = 0.64
2, Test Accuracy = 0.811
Iteration 33, Train Loss = 0.271, Train Accuracy = 0.900, Test Loss = 0.47
0, Test Accuracy = 0.850
Iteration 34, Train Loss = 0.263, Train Accuracy = 0.902, Test Loss = 0.47
9, Test Accuracy = 0.858
Iteration 35, Train Loss = 0.362, Train Accuracy = 0.868, Test Loss = 0.61
2, Test Accuracy = 0.812
Iteration 36, Train Loss = 0.289, Train Accuracy = 0.894, Test Loss = 0.54
0, Test Accuracy = 0.841
Iteration 37, Train Loss = 0.263, Train Accuracy = 0.902, Test Loss = 0.52
2, Test Accuracy = 0.858
Iteration 38, Train Loss = 0.291, Train Accuracy = 0.895, Test Loss = 0.57
1, Test Accuracy = 0.843
Iteration 39, Train Loss = 0.336, Train Accuracy = 0.873, Test Loss = 0.60
9, Test Accuracy = 0.820
Iteration 40, Train Loss = 0.265, Train Accuracy = 0.904, Test Loss = 0.55
2, Test Accuracy = 0.853
Iteration 41, Train Loss = 0.237, Train Accuracy = 0.916, Test Loss = 0.55
3, Test Accuracy = 0.866
Iteration 42, Train Loss = 0.459, Train Accuracy = 0.851, Test Loss = 0.73
7, Test Accuracy = 0.809
Iteration 43, Train Loss = 0.270, Train Accuracy = 0.899, Test Loss = 0.56
4, Test Accuracy = 0.845
Iteration 44, Train Loss = 0.354, Train Accuracy = 0.863, Test Loss = 0.64
5, Test Accuracy = 0.810
Iteration 45, Train Loss = 0.203, Train Accuracy = 0.928, Test Loss = 0.52
6, Test Accuracy = 0.869
Iteration 46, Train Loss = 0.292, Train Accuracy = 0.898, Test Loss = 0.61
1, Test Accuracy = 0.831
Iteration 47, Train Loss = 0.202, Train Accuracy = 0.927, Test Loss = 0.51
1, Test Accuracy = 0.866
Iteration 48, Train Loss = 0.383, Train Accuracy = 0.863, Test Loss = 0.67
5, Test Accuracy = 0.808
Iteration 49, Train Loss = 0.338, Train Accuracy = 0.875, Test Loss = 0.63
3, Test Accuracy = 0.815
Iteration 50, Train Loss = 0.440, Train Accuracy = 0.844, Test Loss = 0.75
1, Test Accuracy = 0.798
Iteration 51, Train Loss = 0.240, Train Accuracy = 0.914, Test Loss = 0.53
5, Test Accuracy = 0.853
Iteration 52, Train Loss = 0.288, Train Accuracy = 0.896, Test Loss = 0.57
6, Test Accuracy = 0.841
Iteration 53, Train Loss = 0.300, Train Accuracy = 0.890, Test Loss = 0.60
6, Test Accuracy = 0.831
Iteration 54, Train Loss = 0.228, Train Accuracy = 0.919, Test Loss = 0.55
0, Test Accuracy = 0.858
Iteration 55, Train Loss = 0.225, Train Accuracy = 0.917, Test Loss = 0.52
1, Test Accuracy = 0.857
Iteration 56, Train Loss = 0.372, Train Accuracy = 0.871, Test Loss = 0.71
7, Test Accuracy = 0.814
```

```
Iteration 57, Train Loss = 0.328, Train Accuracy = 0.883, Test Loss = 0.68
1, Test Accuracy = 0.821
Iteration 58, Train Loss = 0.319, Train Accuracy = 0.898, Test Loss = 0.67
6, Test Accuracy = 0.840
Iteration 59, Train Loss = 0.195, Train Accuracy = 0.926, Test Loss = 0.51
9, Test Accuracy = 0.865
Iteration 60, Train Loss = 0.315, Train Accuracy = 0.888, Test Loss = 0.65
2, Test Accuracy = 0.839
Iteration 61, Train Loss = 0.318, Train Accuracy = 0.886, Test Loss = 0.65
0, Test Accuracy = 0.831
Iteration 62, Train Loss = 0.233, Train Accuracy = 0.917, Test Loss = 0.60
4, Test Accuracy = 0.848
Iteration 63, Train Loss = 0.358, Train Accuracy = 0.875, Test Loss = 0.70
6, Test Accuracy = 0.813
Iteration 64, Train Loss = 0.281, Train Accuracy = 0.888, Test Loss = 0.62
7, Test Accuracy = 0.826
Iteration 65, Train Loss = 0.219, Train Accuracy = 0.924, Test Loss = 0.60
0, Test Accuracy = 0.854
Iteration 66, Train Loss = 0.311, Train Accuracy = 0.884, Test Loss = 0.67
0, Test Accuracy = 0.825
Iteration 67, Train Loss = 0.412, Train Accuracy = 0.847, Test Loss = 0.69
4, Test Accuracy = 0.800
Iteration 68, Train Loss = 0.338, Train Accuracy = 0.885, Test Loss = 0.72
5, Test Accuracy = 0.819
Iteration 69, Train Loss = 0.233, Train Accuracy = 0.919, Test Loss = 0.67
0, Test Accuracy = 0.842
Iteration 70, Train Loss = 0.293, Train Accuracy = 0.907, Test Loss = 0.74
5, Test Accuracy = 0.834
Iteration 71, Train Loss = 0.347, Train Accuracy = 0.878, Test Loss = 0.71
5, Test Accuracy = 0.822
Iteration 72, Train Loss = 0.304, Train Accuracy = 0.899, Test Loss = 0.74
8, Test Accuracy = 0.834
Iteration 73, Train Loss = 0.387, Train Accuracy = 0.876, Test Loss = 0.77
5, Test Accuracy = 0.819
Iteration 74, Train Loss = 0.294, Train Accuracy = 0.897, Test Loss = 0.68
8, Test Accuracy = 0.830
Iteration 75, Train Loss = 0.345, Train Accuracy = 0.874, Test Loss = 0.72
4, Test Accuracy = 0.806
Iteration 76, Train Loss = 0.360, Train Accuracy = 0.881, Test Loss = 0.73
8, Test Accuracy = 0.824
Iteration 77, Train Loss = 0.138, Train Accuracy = 0.951, Test Loss = 0.51
5, Test Accuracy = 0.875
Iteration 78, Train Loss = 0.187, Train Accuracy = 0.932, Test Loss = 0.58
6, Test Accuracy = 0.859
Iteration 79, Train Loss = 0.188, Train Accuracy = 0.930, Test Loss = 0.61
4, Test Accuracy = 0.856
Iteration 80, Train Loss = 0.343, Train Accuracy = 0.884, Test Loss = 0.77
9, Test Accuracy = 0.813
Iteration 81, Train Loss = 0.155, Train Accuracy = 0.944, Test Loss = 0.54
7, Test Accuracy = 0.863
Iteration 82, Train Loss = 0.286, Train Accuracy = 0.892, Test Loss = 0.63
0, Test Accuracy = 0.819
Iteration 83, Train Loss = 0.217, Train Accuracy = 0.924, Test Loss = 0.56
2, Test Accuracy = 0.854
Iteration 84, Train Loss = 0.290, Train Accuracy = 0.900, Test Loss = 0.67
9, Test Accuracy = 0.826
```

```
Iteration 85, Train Loss = 0.206, Train Accuracy = 0.926, Test Loss = 0.62
5, Test Accuracy = 0.848
Iteration 86, Train Loss = 0.196, Train Accuracy = 0.932, Test Loss = 0.57
1, Test Accuracy = 0.859
Iteration 87, Train Loss = 0.318, Train Accuracy = 0.897, Test Loss = 0.72
1, Test Accuracy = 0.829
Iteration 88, Train Loss = 0.276, Train Accuracy = 0.900, Test Loss = 0.66
5, Test Accuracy = 0.834
Iteration 89, Train Loss = 0.217, Train Accuracy = 0.918, Test Loss = 0.64
8, Test Accuracy = 0.840
Iteration 90, Train Loss = 0.179, Train Accuracy = 0.935, Test Loss = 0.64
4, Test Accuracy = 0.856
Iteration 91, Train Loss = 0.295, Train Accuracy = 0.907, Test Loss = 0.80
3, Test Accuracy = 0.833
Iteration 92, Train Loss = 0.162, Train Accuracy = 0.940, Test Loss = 0.58
2, Test Accuracy = 0.855
Iteration 93, Train Loss = 0.351, Train Accuracy = 0.870, Test Loss = 0.80
7, Test Accuracy = 0.798
Iteration 94, Train Loss = 0.215, Train Accuracy = 0.923, Test Loss = 0.67
5, Test Accuracy = 0.838
Iteration 95, Train Loss = 0.192, Train Accuracy = 0.931, Test Loss = 0.64
5, Test Accuracy = 0.845
Iteration 96, Train Loss = 0.191, Train Accuracy = 0.930, Test Loss = 0.63
5, Test Accuracy = 0.850
Iteration 97, Train Loss = 0.208, Train Accuracy = 0.920, Test Loss = 0.66
0, Test Accuracy = 0.843
Iteration 98, Train Loss = 0.130, Train Accuracy = 0.950, Test Loss = 0.60
4, Test Accuracy = 0.859
Iteration 99, Train Loss = 0.140, Train Accuracy = 0.950, Test Loss = 0.64
9, Test Accuracy = 0.859
Iteration 100, Train Loss = 0.173, Train Accuracy = 0.935, Test Loss = 0.52
7, Test Accuracy = 0.861
```

2.4 Test data with augmentations

In [ ]:

```
@torch.no_grad()
def compute_accuracy_augment(net, X, y):
    """ Compute the classification accuracy of the augmented set
        X of shape (n, d) and y of shape (n,)
    """
    pooled_preds = torch.zeros(y.shape[0], 8)
    for i in range(8):
        X_augment = transform_selected_data(X)
        score = net(X_augment)
        predictions = torch.argmax(score, axis=1)  # class with highest scor
        pooled_preds[:,i] = predictions

    final_predictions = torch.mode(pooled_preds, 1).values
    # Return the fraction of predictions that are correct
    return (final_predictions == y).sum() * 1.0 / y.shape[0]

@torch.no_grad()
def compute_logs_test_augment(net, verbose=False):
    train_loss = compute_objective(net, X_train, y_train)
    train_accuracy = compute_accuracy(net, X_train, y_train)
    test_loss = compute_objective(net, X_test, y_test)
    test_accuracy = compute_accuracy_augment(net, X_test, y_test)
    if verbose:
        print(('Train Loss = {:.3f}, Train Accuracy = {:.3f}, ' +
               'Test Loss = {:.3f}, Test Accuracy = {:.3f}').format(
                train_loss.item(), train_accuracy.item(),
                test_loss.item(), test_accuracy.item())
    )
    return (train_loss, train_accuracy, test_loss, test_accuracy)
```

In [ ]:

```
learning_rate = 0.04

logs_3 = []

model = MyConvNet(num_classes=10)
print('Iteration 0', end=', ')
logs_3.append(compute_logs_test_augment(model, verbose=True))

batch_size = 16

for j in range(100):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs_test_augment(model, verbose=True)
    logs_3.append(log)
    if log[1] == 1.0:
        break
```

```
Iteration 0, Train Loss = 2.310, Train Accuracy = 0.059, Test Loss = 2.312,
Test Accuracy = 0.077
Iteration 1, Train Loss = 0.482, Train Accuracy = 0.831, Test Loss = 0.530,
Test Accuracy = 0.786
Iteration 2, Train Loss = 0.389, Train Accuracy = 0.861, Test Loss = 0.474,
Test Accuracy = 0.793
Iteration 3, Train Loss = 0.356, Train Accuracy = 0.873, Test Loss = 0.460,
Test Accuracy = 0.806
Iteration 4, Train Loss = 0.312, Train Accuracy = 0.891, Test Loss = 0.444,
Test Accuracy = 0.812
Iteration 5, Train Loss = 0.286, Train Accuracy = 0.897, Test Loss = 0.437,
Test Accuracy = 0.806
Iteration 6, Train Loss = 0.278, Train Accuracy = 0.898, Test Loss = 0.470,
Test Accuracy = 0.819
Iteration 7, Train Loss = 0.219, Train Accuracy = 0.925, Test Loss = 0.469,
Test Accuracy = 0.808
Iteration 8, Train Loss = 0.178, Train Accuracy = 0.939, Test Loss = 0.416,
Test Accuracy = 0.826
Iteration 9, Train Loss = 0.159, Train Accuracy = 0.944, Test Loss = 0.444,
Test Accuracy = 0.841
Iteration 10, Train Loss = 0.154, Train Accuracy = 0.946, Test Loss = 0.48
2, Test Accuracy = 0.813
Iteration 11, Train Loss = 0.125, Train Accuracy = 0.959, Test Loss = 0.49
1, Test Accuracy = 0.830
Iteration 12, Train Loss = 0.112, Train Accuracy = 0.963, Test Loss = 0.51
7, Test Accuracy = 0.837
Iteration 13, Train Loss = 0.122, Train Accuracy = 0.958, Test Loss = 0.57
0, Test Accuracy = 0.850
Iteration 14, Train Loss = 0.077, Train Accuracy = 0.975, Test Loss = 0.54
4, Test Accuracy = 0.833
Iteration 15, Train Loss = 0.080, Train Accuracy = 0.972, Test Loss = 0.54
2, Test Accuracy = 0.848
Iteration 16, Train Loss = 0.080, Train Accuracy = 0.972, Test Loss = 0.57
8, Test Accuracy = 0.852
Iteration 17, Train Loss = 0.080, Train Accuracy = 0.974, Test Loss = 0.62
5, Test Accuracy = 0.834
Iteration 18, Train Loss = 0.076, Train Accuracy = 0.974, Test Loss = 0.68
2, Test Accuracy = 0.829
Iteration 19, Train Loss = 0.047, Train Accuracy = 0.985, Test Loss = 0.68
2, Test Accuracy = 0.833
Iteration 20, Train Loss = 0.045, Train Accuracy = 0.986, Test Loss = 0.70
0, Test Accuracy = 0.843
Iteration 21, Train Loss = 0.028, Train Accuracy = 0.992, Test Loss = 0.68
5, Test Accuracy = 0.854
Iteration 22, Train Loss = 0.056, Train Accuracy = 0.981, Test Loss = 0.74
4, Test Accuracy = 0.851
Iteration 23, Train Loss = 0.031, Train Accuracy = 0.991, Test Loss = 0.69
3, Test Accuracy = 0.821
Iteration 24, Train Loss = 0.039, Train Accuracy = 0.988, Test Loss = 0.75
4, Test Accuracy = 0.847
Iteration 25, Train Loss = 0.015, Train Accuracy = 0.997, Test Loss = 0.74
2, Test Accuracy = 0.816
Iteration 26, Train Loss = 0.011, Train Accuracy = 0.998, Test Loss = 0.75
4, Test Accuracy = 0.828
Iteration 27, Train Loss = 0.009, Train Accuracy = 0.998, Test Loss = 0.79
7, Test Accuracy = 0.851
```

```
Iteration 28, Train Loss = 0.005, Train Accuracy = 1.000, Test Loss = 0.79
7, Test Accuracy = 0.857
Iteration 29, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 0.81
6, Test Accuracy = 0.852
Iteration 30, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 0.83
0, Test Accuracy = 0.831
```

2.5 Train and test data with augmentations

In [ ]:
```
learning_rate = 0.04

logs_4 = []

model = MyConvNet(num_classes=10)
print('Iteration 0', end=', ')
logs_4.append(compute_logs_test_augment(model, verbose=True))

batch_size = 16

for j in range(100):
    X_train_augmented = transform_selected_data(X_train)
    model = minibatch_sgd_one_pass(model, X_train_augmented, y_train, learni
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs_test_augment(model, verbose=True)
    logs_4.append(log)
    if log[1] == 1.0:
        break
```

```
Iteration 0, Train Loss = 2.332, Train Accuracy = 0.090, Test Loss = 2.330,
Test Accuracy = 0.103
Iteration 1, Train Loss = 0.860, Train Accuracy = 0.698, Test Loss = 0.889,
Test Accuracy = 0.721
Iteration 2, Train Loss = 0.582, Train Accuracy = 0.792, Test Loss = 0.635,
Test Accuracy = 0.733
Iteration 3, Train Loss = 0.736, Train Accuracy = 0.725, Test Loss = 0.814,
Test Accuracy = 0.718
Iteration 4, Train Loss = 0.464, Train Accuracy = 0.842, Test Loss = 0.541,
Test Accuracy = 0.806
Iteration 5, Train Loss = 0.618, Train Accuracy = 0.798, Test Loss = 0.711,
Test Accuracy = 0.776
Iteration 6, Train Loss = 0.392, Train Accuracy = 0.860, Test Loss = 0.498,
Test Accuracy = 0.834
Iteration 7, Train Loss = 0.572, Train Accuracy = 0.832, Test Loss = 0.686,
Test Accuracy = 0.814
Iteration 8, Train Loss = 0.431, Train Accuracy = 0.842, Test Loss = 0.531,
Test Accuracy = 0.819
Iteration 9, Train Loss = 0.428, Train Accuracy = 0.848, Test Loss = 0.576,
Test Accuracy = 0.807
Iteration 10, Train Loss = 0.400, Train Accuracy = 0.855, Test Loss = 0.49
5, Test Accuracy = 0.800
Iteration 11, Train Loss = 0.379, Train Accuracy = 0.861, Test Loss = 0.53
1, Test Accuracy = 0.843
Iteration 12, Train Loss = 0.411, Train Accuracy = 0.859, Test Loss = 0.59
1, Test Accuracy = 0.806
Iteration 13, Train Loss = 0.273, Train Accuracy = 0.902, Test Loss = 0.44
0, Test Accuracy = 0.843
Iteration 14, Train Loss = 0.348, Train Accuracy = 0.878, Test Loss = 0.50
7, Test Accuracy = 0.834
Iteration 15, Train Loss = 0.351, Train Accuracy = 0.875, Test Loss = 0.49
5, Test Accuracy = 0.843
Iteration 16, Train Loss = 0.514, Train Accuracy = 0.834, Test Loss = 0.67
9, Test Accuracy = 0.807
Iteration 17, Train Loss = 0.353, Train Accuracy = 0.870, Test Loss = 0.49
3, Test Accuracy = 0.830
Iteration 18, Train Loss = 0.256, Train Accuracy = 0.909, Test Loss = 0.42
6, Test Accuracy = 0.839
Iteration 19, Train Loss = 0.344, Train Accuracy = 0.876, Test Loss = 0.50
3, Test Accuracy = 0.854
Iteration 20, Train Loss = 0.294, Train Accuracy = 0.894, Test Loss = 0.48
7, Test Accuracy = 0.850
Iteration 21, Train Loss = 0.285, Train Accuracy = 0.898, Test Loss = 0.47
8, Test Accuracy = 0.793
Iteration 22, Train Loss = 0.326, Train Accuracy = 0.883, Test Loss = 0.52
4, Test Accuracy = 0.845
Iteration 23, Train Loss = 0.298, Train Accuracy = 0.891, Test Loss = 0.47
6, Test Accuracy = 0.869
Iteration 24, Train Loss = 0.405, Train Accuracy = 0.847, Test Loss = 0.58
1, Test Accuracy = 0.782
Iteration 25, Train Loss = 0.343, Train Accuracy = 0.869, Test Loss = 0.55
5, Test Accuracy = 0.833
Iteration 26, Train Loss = 0.336, Train Accuracy = 0.879, Test Loss = 0.54
9, Test Accuracy = 0.832
Iteration 27, Train Loss = 0.320, Train Accuracy = 0.882, Test Loss = 0.55
9, Test Accuracy = 0.841
```

```
Iteration 28, Train Loss = 0.291, Train Accuracy = 0.888, Test Loss = 0.50
5, Test Accuracy = 0.852
Iteration 29, Train Loss = 0.369, Train Accuracy = 0.872, Test Loss = 0.56
8, Test Accuracy = 0.845
Iteration 30, Train Loss = 0.579, Train Accuracy = 0.788, Test Loss = 0.73
0, Test Accuracy = 0.738
Iteration 31, Train Loss = 0.491, Train Accuracy = 0.841, Test Loss = 0.68
1, Test Accuracy = 0.815
Iteration 32, Train Loss = 0.227, Train Accuracy = 0.922, Test Loss = 0.45
6, Test Accuracy = 0.853
Iteration 33, Train Loss = 0.316, Train Accuracy = 0.882, Test Loss = 0.53
7, Test Accuracy = 0.831
Iteration 34, Train Loss = 0.279, Train Accuracy = 0.899, Test Loss = 0.53
1, Test Accuracy = 0.845
Iteration 35, Train Loss = 0.290, Train Accuracy = 0.892, Test Loss = 0.52
4, Test Accuracy = 0.836
Iteration 36, Train Loss = 0.383, Train Accuracy = 0.856, Test Loss = 0.60
2, Test Accuracy = 0.808
Iteration 37, Train Loss = 0.250, Train Accuracy = 0.908, Test Loss = 0.52
8, Test Accuracy = 0.818
Iteration 38, Train Loss = 0.249, Train Accuracy = 0.909, Test Loss = 0.51
8, Test Accuracy = 0.842
Iteration 39, Train Loss = 0.231, Train Accuracy = 0.918, Test Loss = 0.55
5, Test Accuracy = 0.865
Iteration 40, Train Loss = 0.329, Train Accuracy = 0.884, Test Loss = 0.64
3, Test Accuracy = 0.834
Iteration 41, Train Loss = 0.342, Train Accuracy = 0.876, Test Loss = 0.61
5, Test Accuracy = 0.848
Iteration 42, Train Loss = 0.184, Train Accuracy = 0.932, Test Loss = 0.47
4, Test Accuracy = 0.836
Iteration 43, Train Loss = 0.271, Train Accuracy = 0.902, Test Loss = 0.53
7, Test Accuracy = 0.851
Iteration 44, Train Loss = 0.493, Train Accuracy = 0.815, Test Loss = 0.72
2, Test Accuracy = 0.778
Iteration 45, Train Loss = 0.305, Train Accuracy = 0.889, Test Loss = 0.59
3, Test Accuracy = 0.853
Iteration 46, Train Loss = 0.345, Train Accuracy = 0.882, Test Loss = 0.64
8, Test Accuracy = 0.843
Iteration 47, Train Loss = 0.279, Train Accuracy = 0.897, Test Loss = 0.59
7, Test Accuracy = 0.859
Iteration 48, Train Loss = 0.318, Train Accuracy = 0.887, Test Loss = 0.60
7, Test Accuracy = 0.825
Iteration 49, Train Loss = 0.291, Train Accuracy = 0.895, Test Loss = 0.59
8, Test Accuracy = 0.844
Iteration 50, Train Loss = 0.234, Train Accuracy = 0.915, Test Loss = 0.55
8, Test Accuracy = 0.863
Iteration 51, Train Loss = 0.377, Train Accuracy = 0.858, Test Loss = 0.71
4, Test Accuracy = 0.817
Iteration 52, Train Loss = 0.280, Train Accuracy = 0.897, Test Loss = 0.56
3, Test Accuracy = 0.853
Iteration 53, Train Loss = 0.213, Train Accuracy = 0.923, Test Loss = 0.55
3, Test Accuracy = 0.863
Iteration 54, Train Loss = 0.218, Train Accuracy = 0.924, Test Loss = 0.58
6, Test Accuracy = 0.869
Iteration 55, Train Loss = 0.261, Train Accuracy = 0.905, Test Loss = 0.54
7, Test Accuracy = 0.845
```

```
Iteration 56, Train Loss = 0.247, Train Accuracy = 0.908, Test Loss = 0.58
9, Test Accuracy = 0.859
Iteration 57, Train Loss = 0.329, Train Accuracy = 0.882, Test Loss = 0.63
8, Test Accuracy = 0.837
Iteration 58, Train Loss = 0.325, Train Accuracy = 0.884, Test Loss = 0.60
4, Test Accuracy = 0.843
Iteration 59, Train Loss = 0.251, Train Accuracy = 0.906, Test Loss = 0.55
9, Test Accuracy = 0.850
Iteration 60, Train Loss = 0.275, Train Accuracy = 0.901, Test Loss = 0.57
3, Test Accuracy = 0.859
Iteration 61, Train Loss = 0.397, Train Accuracy = 0.858, Test Loss = 0.74
8, Test Accuracy = 0.833
Iteration 62, Train Loss = 0.258, Train Accuracy = 0.907, Test Loss = 0.58
3, Test Accuracy = 0.862
Iteration 63, Train Loss = 0.227, Train Accuracy = 0.916, Test Loss = 0.60
4, Test Accuracy = 0.855
Iteration 64, Train Loss = 0.256, Train Accuracy = 0.904, Test Loss = 0.58
0, Test Accuracy = 0.848
Iteration 65, Train Loss = 0.346, Train Accuracy = 0.876, Test Loss = 0.71
4, Test Accuracy = 0.849
Iteration 66, Train Loss = 0.251, Train Accuracy = 0.911, Test Loss = 0.57
5, Test Accuracy = 0.859
Iteration 67, Train Loss = 0.330, Train Accuracy = 0.877, Test Loss = 0.67
3, Test Accuracy = 0.826
Iteration 68, Train Loss = 0.286, Train Accuracy = 0.892, Test Loss = 0.69
4, Test Accuracy = 0.856
Iteration 69, Train Loss = 0.385, Train Accuracy = 0.870, Test Loss = 0.76
9, Test Accuracy = 0.839
Iteration 70, Train Loss = 0.355, Train Accuracy = 0.868, Test Loss = 0.71
0, Test Accuracy = 0.847
Iteration 71, Train Loss = 0.450, Train Accuracy = 0.851, Test Loss = 0.86
3, Test Accuracy = 0.809
Iteration 72, Train Loss = 0.374, Train Accuracy = 0.866, Test Loss = 0.67
5, Test Accuracy = 0.805
Iteration 73, Train Loss = 0.408, Train Accuracy = 0.856, Test Loss = 0.72
2, Test Accuracy = 0.816
Iteration 74, Train Loss = 0.199, Train Accuracy = 0.925, Test Loss = 0.50
8, Test Accuracy = 0.856
Iteration 75, Train Loss = 0.271, Train Accuracy = 0.904, Test Loss = 0.63
8, Test Accuracy = 0.857
Iteration 76, Train Loss = 0.297, Train Accuracy = 0.896, Test Loss = 0.65
9, Test Accuracy = 0.854
Iteration 77, Train Loss = 0.315, Train Accuracy = 0.881, Test Loss = 0.62
9, Test Accuracy = 0.853
Iteration 78, Train Loss = 0.323, Train Accuracy = 0.884, Test Loss = 0.69
6, Test Accuracy = 0.854
Iteration 79, Train Loss = 0.463, Train Accuracy = 0.849, Test Loss = 0.85
9, Test Accuracy = 0.813
Iteration 80, Train Loss = 0.152, Train Accuracy = 0.942, Test Loss = 0.56
9, Test Accuracy = 0.849
Iteration 81, Train Loss = 0.276, Train Accuracy = 0.896, Test Loss = 0.69
3, Test Accuracy = 0.841
Iteration 82, Train Loss = 0.306, Train Accuracy = 0.898, Test Loss = 0.76
5, Test Accuracy = 0.851
Iteration 83, Train Loss = 0.255, Train Accuracy = 0.910, Test Loss = 0.69
5, Test Accuracy = 0.854
```

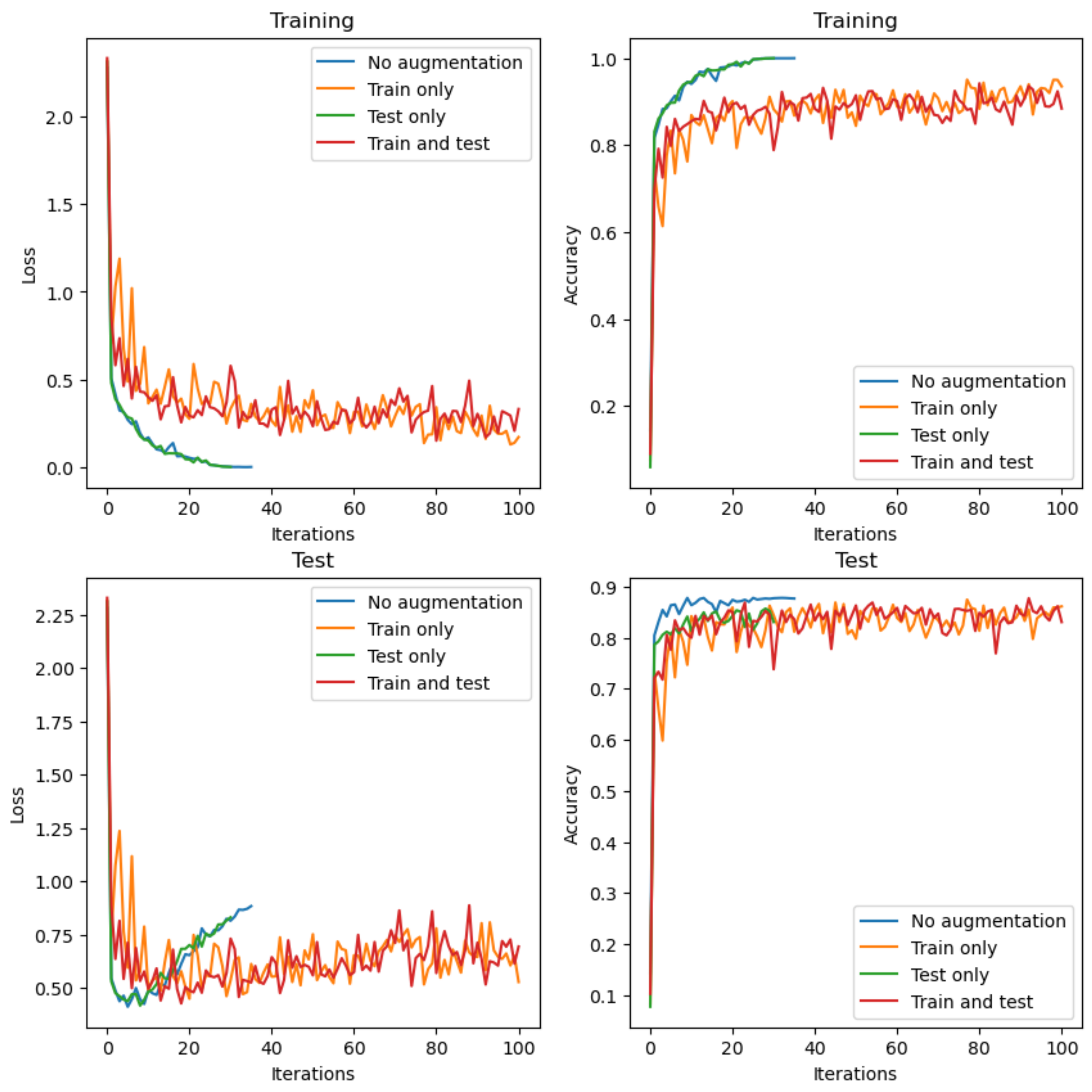
```
Iteration 84, Train Loss = 0.322, Train Accuracy = 0.882, Test Loss = 0.65
3, Test Accuracy = 0.769
Iteration 85, Train Loss = 0.320, Train Accuracy = 0.877, Test Loss = 0.66
8, Test Accuracy = 0.829
Iteration 86, Train Loss = 0.258, Train Accuracy = 0.900, Test Loss = 0.60
5, Test Accuracy = 0.840
Iteration 87, Train Loss = 0.352, Train Accuracy = 0.874, Test Loss = 0.67
6, Test Accuracy = 0.828
Iteration 88, Train Loss = 0.494, Train Accuracy = 0.847, Test Loss = 0.88
7, Test Accuracy = 0.830
Iteration 89, Train Loss = 0.235, Train Accuracy = 0.914, Test Loss = 0.61
4, Test Accuracy = 0.863
Iteration 90, Train Loss = 0.305, Train Accuracy = 0.890, Test Loss = 0.71
2, Test Accuracy = 0.847
Iteration 91, Train Loss = 0.245, Train Accuracy = 0.908, Test Loss = 0.63
6, Test Accuracy = 0.843
Iteration 92, Train Loss = 0.172, Train Accuracy = 0.936, Test Loss = 0.51
5, Test Accuracy = 0.877
Iteration 93, Train Loss = 0.194, Train Accuracy = 0.928, Test Loss = 0.62
4, Test Accuracy = 0.852
Iteration 94, Train Loss = 0.290, Train Accuracy = 0.897, Test Loss = 0.61
8, Test Accuracy = 0.848
Iteration 95, Train Loss = 0.197, Train Accuracy = 0.925, Test Loss = 0.60
4, Test Accuracy = 0.862
Iteration 96, Train Loss = 0.320, Train Accuracy = 0.889, Test Loss = 0.71
9, Test Accuracy = 0.842
Iteration 97, Train Loss = 0.308, Train Accuracy = 0.888, Test Loss = 0.69
2, Test Accuracy = 0.834
Iteration 98, Train Loss = 0.295, Train Accuracy = 0.895, Test Loss = 0.72
6, Test Accuracy = 0.852
Iteration 99, Train Loss = 0.208, Train Accuracy = 0.924, Test Loss = 0.61
3, Test Accuracy = 0.862
Iteration 100, Train Loss = 0.332, Train Accuracy = 0.884, Test Loss = 0.69
3, Test Accuracy = 0.831
```

2.6. Deliverables

In [ ]:
```
print(f"Final test accuracy for - \n1. No augmentations:{round(logs[-1][3].i
3. Test augmentations:{round(logs_3[-1][2].item(),4)}\n4. Train and test aug
```

```
Final test accuracy for -
1. No augmentations:0.8767
2. Training augmentations:0.8613
3. Test augmentations:0.8301
4. Train and test augmentations:0.6932
```

In [ ]:
```
fig, ax = plt.subplots(2,2)
fig.set_size_inches(10,10)
for a in ax.flatten(): a.set_xlabel("Iterations")
for a in ax: a[0].set_ylabel("Loss")
for a in ax: a[1].set_ylabel("Accuracy")
for a in ax[0,:]: a.set_title("Training")
for a in ax[1,:]: a.set_title("Test")

names = zip([logs, logs_2, logs_3, logs_4], ["No augmentation", "Train only"
for log, label in names:
    log = np.asarray(log)
    ax[0,0].plot(log[:,0], label = label)
    ax[0,0].legend()
    ax[0,1].plot(log[:,1], label = label)
    ax[0,1].legend()
    ax[1,0].plot(log[:,2], label = label)
    ax[1,0].legend()
    ax[1,1].plot(log[:,3], label = label)
    ax[1,1].legend()
```



3. (Bonus) Serial updates and gradient computations with minibatching

In this exercise, we will experiment with minibatching in a multilayer perceptron (MLP) with a single hidden layer. When we make N passes over the data with n points, we have a total of Nn gradient computations. However, the number of updates we make to our model are:

- SGD: nN updates
- Minibatch SGD (batch size B): ⌊nN/B⌋

On the other hand, if we make T updates to our model, the number of gradient computations we need are:

- SGD: T gradient computations
- Minibatch SGD (batch size B): TB gradient computations.

Sometimes, the number of updates is the bottleneck (that is, computing the minibatch stochastic gradient is not much more expensive that computing a stochastic gradient with B = 1). This is true, for instance, on a GPU or similar hardware which allow massive parallelism. We will explore this setting in this exercise.

Here are the details:

- The setup is identical to the lab. Take the FashionMNIST dataset and randomly subsample 10% of its training set to work with. As a test set, we will use the full test set of FashionMNIST.
- Define a MLP with one hidden layer of width h = 64. This model stays fixed throughout the homework.
- For each batch size B in [1, 2, 4, 8, 16], find the divergent learning rate ηB∗ . Use a fixed learning rate of ηB∗ /2 for this batch size.
- Train the model for 50n/B total updates,where n is the number of training examples. This corresponds to 50 passes over the data with a batch size of B.

The deliverables for this exercise are:

1. Make 4 plots, one each for the train loss, train accuracy, test loss and test accuracy over the course of training (i.e., the metric on the y-axis and number of updates on the x-axis). Plot all 5 lines, one for each value of B on the same plot.
2. When the training accuracy is 100%, the model is said to interpolate the training data. As we vary the batch size of the network, after how many updates do we observe perfect interpolation of the data? That is, make a plot with B on the x-axis and number of updates over the data required for interpolation on the y axis.

3.1 Model and dataset setup

Loading functions that will be required for model training and setting up the dataset.

In [ ]:
```
train_dataset = FashionMNIST('../../data', train=True, download=False)
X_train = train_dataset.data # torch tensor of type uint8 of shape (n, 28, 2
y_train = train_dataset.targets.long() # torch tensor of type Long of shape
```

```
test_dataset = FashionMNIST('../../data', train=False, download=False)
X_test = test_dataset.data
y_test = test_dataset.targets.long()

# choose a subsample of 10% of the data:
idxs_train = torch.from_numpy(
    np.random.choice(X_train.shape[0], replace=False, size=X_train.shape[0]/
X_train, y_train = X_train[idxs_train], y_train[idxs_train]
idxs_test = torch.from_numpy(
    np.random.choice(X_test.shape[0], replace=False, size=X_test.shape[0]))
X_test, y_test = X_test[idxs_test], y_test[idxs_test]

print(f'X_train.shape = {X_train.shape}')
print(f'n_train: {X_train.shape[0]}, n_test: {X_test.shape[0]}')
print(f'Image size: {X_train.shape[1:]}')
```

```
X_train.shape = torch.Size([6000, 28, 28])
n_train: 6000, n_test: 10000
Image size: torch.Size([28, 28])
```

In [ ]:
```
print("Original Shape:", X_train.shape)
X_train = X_train.float()  # convert to float32. Shape: (n, 28, 28)
X_train = X_train.view(-1, 784)  # Shape: (n, 784)
print("Flatten Shape:", X_train.shape)
mean, std = X_train.mean(axis=0), X_train.std(axis=0)  # Shape: (784,)
```

```
Original Shape: torch.Size([6000, 28, 28])
Flatten Shape: torch.Size([6000, 784])
```

In [ ]:
```
# Normalize dataset: pixel values lie between 0 and 255
# Normalize them so the pixelwise mean is zero and standard deviation is 1
# Normalize: add a small number to avoid divide by zero
X_train = (X_train - mean[None, :]) / (std[None, :] + 1e-6) # Shape: (n, 784

X_test = X_test.float()  # Shape: (n', 28, 28)
X_test = X_test.view(-1, 784)  # Shape: (n', 784)
X_test = (X_test - mean[None, :]) / (std[None, :] + 1e-6)  # Shape: (n', 784

n_class = np.unique(y_train).shape[0]  # We have K=10 classes numbered (0, 1
print("Number of Classes: ", n_class)
```

```
Number of Classes:  10
```

In [ ]:
```
class MyMLP(torch.nn.Module):
    def __init__(self, input_size=28*28, hidden_size=64, output_size=10):
        super().__init__()
        self.first = torch.nn.Linear(input_size, hidden_size, bias = True)
        self.relu = torch.nn.ReLU()
        self.hidden = torch.nn.Linear(hidden_size, output_size, bias=True)

    def forward(self, x):
        out = self.first(x)
        out = self.relu(out)
        out = self.hidden(out)
        return out
```

Modifying functions to calculate loss/accuracy at every update instead of every pass of the data

3.2 Batchsize B=1

In [ ]:
```
learning_rate = 1e-2 #This is the divergent learning rate because loss drops
# We will use half of this learning rate to train the model

m = MyMLP()

_ = compute_logs(m, verbose=True)

m = minibatch_sgd_one_pass(m, X_train, y_train, learning_rate, batch_size=1,

_ = compute_logs(m, verbose=True)
```

```
Train Loss = 2.367, Train Accuracy = 0.058, Test Loss = 2.372, Test Accurac
y = 0.056
1.036471576834089
0.931091012188894
0.9125024892709014
0.8399499971678673
0.9214376455446884
0.843004008226712
0.6993131166711072
0.7116826964851756
0.6776413271770011
0.6771319311476256
0.6577198759474848
0.7042133185681928
0.5781498285146264
0.717647847093999
0.7242962873678531
0.6630689073022881
0.5635416413014801
0.7100215144592344
0.6853867540336597
0.7769377214165766
0.6214852416275507
0.5241256024914017
0.6842132063587668
0.7451404740244929
0.7028898226054723
0.7555380327777429
0.5960451654870306
0.4314300076808676
0.5847408068097318
0.6169161980701053
0.511817756886313
0.6011300423267112
0.5656985251248814
0.5563998503790685
0.6071425776611518
0.5268229808761893
0.5572756007134897
0.5785288598446308
0.6119639525774416
0.49064945921548864
0.7361354062750065
0.6212856461153885
0.5817312321811593
0.511605980319583
0.5193560397601062
0.44522069851631957
0.4916144653420903
0.46775712469473346
0.5667961087211021
0.5417235900682095
0.4902459079428062
0.43291459550908906
0.554396502215544
0.4823639719392817
```

```
0.5565353897662957
0.5435723630809108
0.48804922368572007
0.5897636538547472
0.5987048522165253
0.5663305928191696
Train Loss = 0.521, Train Accuracy = 0.824, Test Loss = 0.704, Test Accurac
y = 0.784
```

In [ ]:

```
learning_rate = 5e-3

logs_1 = []

model = MyMLP()
print('Iteration 0', end=', ')
logs_1.append(compute_logs(model, verbose=True))

batch_size = 1

for j in range(50):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_1.append(log)
```

```
Iteration 0, Train Loss = 2.338, Train Accuracy = 0.067, Test Loss = 2.335,
Test Accuracy = 0.067
Iteration 1, Train Loss = 0.453, Train Accuracy = 0.840, Test Loss = 0.576,
Test Accuracy = 0.808
Iteration 2, Train Loss = 0.356, Train Accuracy = 0.870, Test Loss = 0.532,
Test Accuracy = 0.816
Iteration 3, Train Loss = 0.318, Train Accuracy = 0.882, Test Loss = 0.553,
Test Accuracy = 0.817
Iteration 4, Train Loss = 0.284, Train Accuracy = 0.893, Test Loss = 0.561,
Test Accuracy = 0.813
Iteration 5, Train Loss = 0.234, Train Accuracy = 0.914, Test Loss = 0.538,
Test Accuracy = 0.834
Iteration 6, Train Loss = 0.227, Train Accuracy = 0.917, Test Loss = 0.591,
Test Accuracy = 0.819
Iteration 7, Train Loss = 0.215, Train Accuracy = 0.925, Test Loss = 0.613,
Test Accuracy = 0.824
Iteration 8, Train Loss = 0.283, Train Accuracy = 0.905, Test Loss = 0.738,
Test Accuracy = 0.810
Iteration 9, Train Loss = 0.168, Train Accuracy = 0.943, Test Loss = 0.621,
Test Accuracy = 0.829
Iteration 10, Train Loss = 0.170, Train Accuracy = 0.934, Test Loss = 0.67
9, Test Accuracy = 0.820
Iteration 11, Train Loss = 0.147, Train Accuracy = 0.946, Test Loss = 0.69
8, Test Accuracy = 0.826
Iteration 12, Train Loss = 0.107, Train Accuracy = 0.961, Test Loss = 0.66
8, Test Accuracy = 0.841
Iteration 13, Train Loss = 0.141, Train Accuracy = 0.945, Test Loss = 0.76
3, Test Accuracy = 0.825
Iteration 14, Train Loss = 0.096, Train Accuracy = 0.964, Test Loss = 0.72
3, Test Accuracy = 0.837
Iteration 15, Train Loss = 0.079, Train Accuracy = 0.970, Test Loss = 0.75
9, Test Accuracy = 0.838
Iteration 16, Train Loss = 0.062, Train Accuracy = 0.979, Test Loss = 0.73
2, Test Accuracy = 0.842
Iteration 17, Train Loss = 0.069, Train Accuracy = 0.976, Test Loss = 0.79
5, Test Accuracy = 0.838
Iteration 18, Train Loss = 0.058, Train Accuracy = 0.980, Test Loss = 0.80
1, Test Accuracy = 0.841
Iteration 19, Train Loss = 0.071, Train Accuracy = 0.978, Test Loss = 0.85
5, Test Accuracy = 0.834
Iteration 20, Train Loss = 0.075, Train Accuracy = 0.974, Test Loss = 0.89
6, Test Accuracy = 0.840
Iteration 21, Train Loss = 0.104, Train Accuracy = 0.966, Test Loss = 0.97
2, Test Accuracy = 0.825
Iteration 22, Train Loss = 0.032, Train Accuracy = 0.991, Test Loss = 0.85
9, Test Accuracy = 0.841
Iteration 23, Train Loss = 0.045, Train Accuracy = 0.987, Test Loss = 0.93
2, Test Accuracy = 0.834
Iteration 24, Train Loss = 0.042, Train Accuracy = 0.986, Test Loss = 0.95
5, Test Accuracy = 0.837
Iteration 25, Train Loss = 0.024, Train Accuracy = 0.994, Test Loss = 0.94
1, Test Accuracy = 0.841
Iteration 26, Train Loss = 0.017, Train Accuracy = 0.997, Test Loss = 0.93
2, Test Accuracy = 0.842
Iteration 27, Train Loss = 0.011, Train Accuracy = 0.998, Test Loss = 0.94
5, Test Accuracy = 0.841
```

```
Iteration 28, Train Loss = 0.014, Train Accuracy = 0.997, Test Loss = 0.96
7, Test Accuracy = 0.838
Iteration 29, Train Loss = 0.022, Train Accuracy = 0.994, Test Loss = 1.00
3, Test Accuracy = 0.840
Iteration 30, Train Loss = 0.009, Train Accuracy = 0.999, Test Loss = 1.00
3, Test Accuracy = 0.840
Iteration 31, Train Loss = 0.013, Train Accuracy = 0.996, Test Loss = 1.04
3, Test Accuracy = 0.840
Iteration 32, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 1.01
6, Test Accuracy = 0.841
Iteration 33, Train Loss = 0.006, Train Accuracy = 0.999, Test Loss = 1.03
0, Test Accuracy = 0.840
Iteration 34, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 1.03
9, Test Accuracy = 0.839
Iteration 35, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 1.04
3, Test Accuracy = 0.839
Iteration 36, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 1.05
7, Test Accuracy = 0.841
Iteration 37, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.05
9, Test Accuracy = 0.840
Iteration 38, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.07
5, Test Accuracy = 0.840
Iteration 39, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.07
1, Test Accuracy = 0.839
Iteration 40, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.07
6, Test Accuracy = 0.839
Iteration 41, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.08
4, Test Accuracy = 0.840
Iteration 42, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.09
1, Test Accuracy = 0.840
Iteration 43, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.09
6, Test Accuracy = 0.839
Iteration 44, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.09
9, Test Accuracy = 0.840
Iteration 45, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.10
3, Test Accuracy = 0.840
Iteration 46, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.10
5, Test Accuracy = 0.840
Iteration 47, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.10
8, Test Accuracy = 0.841
Iteration 48, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.11
9, Test Accuracy = 0.839
Iteration 49, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.12
1, Test Accuracy = 0.840
Iteration 50, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.12
4, Test Accuracy = 0.840
```

3.3 Batchsize B=2

In [ ]:

```
learning_rate = 2e-2 #This is the divergent learning rate because loss drops
# We will use half of this learning rate to train the model

m = MyMLP()

_ = compute_logs(m, verbose=True)
```

```
m = minibatch_sgd_one_pass(m, X_train, y_train, learning_rate, batch_size=2,

_ = compute_logs(m, verbose=True)
```

```
Train Loss = 2.272, Train Accuracy = 0.185, Test Loss = 2.269, Test Accurac
y = 0.191
0.7445148411100194
0.8077212647425379
0.815522881209398
0.7456174370443162
0.6447236073964702
0.673591367080555
0.6592028608410068
0.6093645437230558
0.5411643899922564
0.5855171564403553
0.6270748145885077
0.6435412921596348
0.7738624741007263
0.8214480197525811
0.7205505732855517
0.6482575724628409
0.6361509394486712
0.6826382215741992
0.5817258602572396
0.6796517066509068
0.6514849047382137
0.6121857436821777
0.5328212852660617
0.6193456958831849
0.5220138127206576
0.5695475433748642
0.5653655491362861
0.49838085524937004
0.562363520225103
0.5797211332038552
Train Loss = 0.565, Train Accuracy = 0.817, Test Loss = 0.736, Test Accurac
y = 0.782
```

In [ ]:
```
learning_rate = 1e-2

logs_2 = []

model = MyMLP()
print('Iteration 0', end=', ')
logs_2.append(compute_logs(model, verbose=True))

batch_size = 2

for j in range(50):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_2.append(log)
```

```
Iteration 0, Train Loss = 2.322, Train Accuracy = 0.145, Test Loss = 2.320,
Test Accuracy = 0.142
Iteration 1, Train Loss = 0.465, Train Accuracy = 0.830, Test Loss = 0.585,
Test Accuracy = 0.795
Iteration 2, Train Loss = 0.346, Train Accuracy = 0.875, Test Loss = 0.517,
Test Accuracy = 0.823
Iteration 3, Train Loss = 0.289, Train Accuracy = 0.895, Test Loss = 0.515,
Test Accuracy = 0.826
Iteration 4, Train Loss = 0.261, Train Accuracy = 0.906, Test Loss = 0.529,
Test Accuracy = 0.828
Iteration 5, Train Loss = 0.239, Train Accuracy = 0.913, Test Loss = 0.555,
Test Accuracy = 0.830
Iteration 6, Train Loss = 0.195, Train Accuracy = 0.931, Test Loss = 0.543,
Test Accuracy = 0.832
Iteration 7, Train Loss = 0.206, Train Accuracy = 0.928, Test Loss = 0.596,
Test Accuracy = 0.828
Iteration 8, Train Loss = 0.153, Train Accuracy = 0.947, Test Loss = 0.570,
Test Accuracy = 0.840
Iteration 9, Train Loss = 0.149, Train Accuracy = 0.947, Test Loss = 0.626,
Test Accuracy = 0.834
Iteration 10, Train Loss = 0.149, Train Accuracy = 0.947, Test Loss = 0.66
2, Test Accuracy = 0.826
Iteration 11, Train Loss = 0.113, Train Accuracy = 0.962, Test Loss = 0.65
0, Test Accuracy = 0.840
Iteration 12, Train Loss = 0.152, Train Accuracy = 0.945, Test Loss = 0.71
8, Test Accuracy = 0.825
Iteration 13, Train Loss = 0.102, Train Accuracy = 0.968, Test Loss = 0.69
7, Test Accuracy = 0.835
Iteration 14, Train Loss = 0.142, Train Accuracy = 0.951, Test Loss = 0.75
1, Test Accuracy = 0.829
Iteration 15, Train Loss = 0.139, Train Accuracy = 0.952, Test Loss = 0.83
1, Test Accuracy = 0.828
Iteration 16, Train Loss = 0.075, Train Accuracy = 0.974, Test Loss = 0.74
6, Test Accuracy = 0.836
Iteration 17, Train Loss = 0.086, Train Accuracy = 0.974, Test Loss = 0.78
8, Test Accuracy = 0.832
Iteration 18, Train Loss = 0.078, Train Accuracy = 0.972, Test Loss = 0.82
4, Test Accuracy = 0.837
Iteration 19, Train Loss = 0.077, Train Accuracy = 0.974, Test Loss = 0.86
9, Test Accuracy = 0.837
Iteration 20, Train Loss = 0.051, Train Accuracy = 0.983, Test Loss = 0.85
3, Test Accuracy = 0.838
Iteration 21, Train Loss = 0.040, Train Accuracy = 0.988, Test Loss = 0.86
3, Test Accuracy = 0.835
Iteration 22, Train Loss = 0.077, Train Accuracy = 0.977, Test Loss = 0.94
1, Test Accuracy = 0.835
Iteration 23, Train Loss = 0.025, Train Accuracy = 0.993, Test Loss = 0.87
8, Test Accuracy = 0.841
Iteration 24, Train Loss = 0.087, Train Accuracy = 0.976, Test Loss = 1.00
7, Test Accuracy = 0.831
Iteration 25, Train Loss = 0.019, Train Accuracy = 0.996, Test Loss = 0.91
7, Test Accuracy = 0.837
Iteration 26, Train Loss = 0.037, Train Accuracy = 0.987, Test Loss = 0.96
5, Test Accuracy = 0.833
Iteration 27, Train Loss = 0.015, Train Accuracy = 0.997, Test Loss = 0.94
2, Test Accuracy = 0.841
```

```
Iteration 28, Train Loss = 0.011, Train Accuracy = 0.998, Test Loss = 0.93
2, Test Accuracy = 0.841
Iteration 29, Train Loss = 0.007, Train Accuracy = 0.999, Test Loss = 0.95
5, Test Accuracy = 0.844
Iteration 30, Train Loss = 0.055, Train Accuracy = 0.980, Test Loss = 1.11
0, Test Accuracy = 0.827
Iteration 31, Train Loss = 0.007, Train Accuracy = 0.999, Test Loss = 0.97
4, Test Accuracy = 0.839
Iteration 32, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.97
7, Test Accuracy = 0.843
Iteration 33, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 1.00
1, Test Accuracy = 0.843
Iteration 34, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 1.00
7, Test Accuracy = 0.843
Iteration 35, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.00
6, Test Accuracy = 0.844
Iteration 36, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.00
8, Test Accuracy = 0.842
Iteration 37, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.01
6, Test Accuracy = 0.844
Iteration 38, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.01
7, Test Accuracy = 0.845
Iteration 39, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.02
4, Test Accuracy = 0.844
Iteration 40, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
1, Test Accuracy = 0.845
Iteration 41, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
5, Test Accuracy = 0.844
Iteration 42, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
9, Test Accuracy = 0.845
Iteration 43, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.04
7, Test Accuracy = 0.844
Iteration 44, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.04
8, Test Accuracy = 0.844
Iteration 45, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.05
2, Test Accuracy = 0.845
Iteration 46, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.05
7, Test Accuracy = 0.845
Iteration 47, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.06
2, Test Accuracy = 0.846
Iteration 48, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.06
7, Test Accuracy = 0.845
Iteration 49, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.06
9, Test Accuracy = 0.845
Iteration 50, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.07
2, Test Accuracy = 0.845
```

3.4 Batchsize B=4

In [ ]:

```
learning_rate = 5e-2 #This is the divergent learning rate because loss drops
# We will use half of this learning rate to train the model

m = MyMLP()

_ = compute_logs(m, verbose=True)
```

```
m = minibatch_sgd_one_pass(m, X_train, y_train, learning_rate, batch_size=4,

_ = compute_logs(m, verbose=True)
```

```
Train Loss = 2.335, Train Accuracy = 0.078, Test Loss = 2.340, Test Accurac
y = 0.076
0.7119482878361818
0.8061354333616219
0.7587156216714426
0.7938728205196014
0.7181763957103294
0.7551026804271102
0.6940284319368567
0.6339157311498452
0.6281401835353183
0.5883750592644067
0.5966305979022258
0.6404588515074932
0.9369026201426864
0.7740291029093868
0.7667293542049956
Train Loss = 0.707, Train Accuracy = 0.786, Test Loss = 0.893, Test Accurac
y = 0.758
```

In [ ]:

```
learning_rate = 2.5e-2

logs_4 = []

model = MyMLP()
print('Iteration 0', end=', ')
logs_4.append(compute_logs(model, verbose=True))

batch_size = 4

for j in range(50):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_4.append(log)
```

```
Iteration 0, Train Loss = 2.335, Train Accuracy = 0.122, Test Loss = 2.334,
Test Accuracy = 0.114
Iteration 1, Train Loss = 0.444, Train Accuracy = 0.844, Test Loss = 0.581,
Test Accuracy = 0.801
Iteration 2, Train Loss = 0.325, Train Accuracy = 0.884, Test Loss = 0.520,
Test Accuracy = 0.825
Iteration 3, Train Loss = 0.314, Train Accuracy = 0.888, Test Loss = 0.558,
Test Accuracy = 0.821
Iteration 4, Train Loss = 0.292, Train Accuracy = 0.894, Test Loss = 0.604,
Test Accuracy = 0.819
Iteration 5, Train Loss = 0.273, Train Accuracy = 0.900, Test Loss = 0.644,
Test Accuracy = 0.809
Iteration 6, Train Loss = 0.230, Train Accuracy = 0.914, Test Loss = 0.603,
Test Accuracy = 0.824
Iteration 7, Train Loss = 0.173, Train Accuracy = 0.938, Test Loss = 0.602,
Test Accuracy = 0.827
Iteration 8, Train Loss = 0.198, Train Accuracy = 0.931, Test Loss = 0.684,
Test Accuracy = 0.824
Iteration 9, Train Loss = 0.178, Train Accuracy = 0.938, Test Loss = 0.705,
Test Accuracy = 0.823
Iteration 10, Train Loss = 0.156, Train Accuracy = 0.948, Test Loss = 0.76
3, Test Accuracy = 0.822
Iteration 11, Train Loss = 0.141, Train Accuracy = 0.947, Test Loss = 0.77
8, Test Accuracy = 0.830
Iteration 12, Train Loss = 0.172, Train Accuracy = 0.943, Test Loss = 0.81
7, Test Accuracy = 0.824
Iteration 13, Train Loss = 0.192, Train Accuracy = 0.934, Test Loss = 0.86
0, Test Accuracy = 0.820
Iteration 14, Train Loss = 0.231, Train Accuracy = 0.921, Test Loss = 0.98
4, Test Accuracy = 0.808
Iteration 15, Train Loss = 0.086, Train Accuracy = 0.972, Test Loss = 0.81
9, Test Accuracy = 0.835
Iteration 16, Train Loss = 0.070, Train Accuracy = 0.975, Test Loss = 0.85
9, Test Accuracy = 0.832
Iteration 17, Train Loss = 0.144, Train Accuracy = 0.948, Test Loss = 1.01
3, Test Accuracy = 0.809
Iteration 18, Train Loss = 0.048, Train Accuracy = 0.984, Test Loss = 0.88
0, Test Accuracy = 0.837
Iteration 19, Train Loss = 0.079, Train Accuracy = 0.970, Test Loss = 0.96
0, Test Accuracy = 0.833
Iteration 20, Train Loss = 0.045, Train Accuracy = 0.986, Test Loss = 0.92
1, Test Accuracy = 0.833
Iteration 21, Train Loss = 0.060, Train Accuracy = 0.979, Test Loss = 1.01
4, Test Accuracy = 0.833
Iteration 22, Train Loss = 0.038, Train Accuracy = 0.988, Test Loss = 0.99
8, Test Accuracy = 0.837
Iteration 23, Train Loss = 0.047, Train Accuracy = 0.982, Test Loss = 1.05
8, Test Accuracy = 0.831
Iteration 24, Train Loss = 0.035, Train Accuracy = 0.989, Test Loss = 1.06
9, Test Accuracy = 0.833
Iteration 25, Train Loss = 0.058, Train Accuracy = 0.979, Test Loss = 1.13
9, Test Accuracy = 0.827
Iteration 26, Train Loss = 0.026, Train Accuracy = 0.992, Test Loss = 1.08
4, Test Accuracy = 0.833
Iteration 27, Train Loss = 0.057, Train Accuracy = 0.981, Test Loss = 1.15
4, Test Accuracy = 0.833
```

```
Iteration 28, Train Loss = 0.027, Train Accuracy = 0.990, Test Loss = 1.12
9, Test Accuracy = 0.836
Iteration 29, Train Loss = 0.017, Train Accuracy = 0.996, Test Loss = 1.13
7, Test Accuracy = 0.835
Iteration 30, Train Loss = 0.022, Train Accuracy = 0.992, Test Loss = 1.16
2, Test Accuracy = 0.837
Iteration 31, Train Loss = 0.013, Train Accuracy = 0.997, Test Loss = 1.14
2, Test Accuracy = 0.835
Iteration 32, Train Loss = 0.036, Train Accuracy = 0.988, Test Loss = 1.21
3, Test Accuracy = 0.824
Iteration 33, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 1.15
0, Test Accuracy = 0.837
Iteration 34, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.16
4, Test Accuracy = 0.838
Iteration 35, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.16
3, Test Accuracy = 0.837
Iteration 36, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.18
5, Test Accuracy = 0.838
Iteration 37, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.18
6, Test Accuracy = 0.839
Iteration 38, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.19
9, Test Accuracy = 0.838
Iteration 39, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.20
9, Test Accuracy = 0.839
Iteration 40, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.20
7, Test Accuracy = 0.839
Iteration 41, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.21
3, Test Accuracy = 0.840
Iteration 42, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.21
7, Test Accuracy = 0.839
Iteration 43, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.22
5, Test Accuracy = 0.838
Iteration 44, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.23
0, Test Accuracy = 0.839
Iteration 45, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.23
3, Test Accuracy = 0.839
Iteration 46, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.23
7, Test Accuracy = 0.840
Iteration 47, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.24
4, Test Accuracy = 0.840
Iteration 48, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.24
4, Test Accuracy = 0.840
Iteration 49, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.24
9, Test Accuracy = 0.839
Iteration 50, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.25
3, Test Accuracy = 0.840
```

3.5 Batchsize B=8

In [ ]:
```
learning_rate = 1e-1 #This is the divergent learning rate because loss drops
# We will use half of this learning rate to train the model

m = MyMLP()

_ = compute_logs(m, verbose=True)
```

```
m = minibatch_sgd_one_pass(m, X_train, y_train, learning_rate, batch_size=8,

_ = compute_logs(m, verbose=True)
```

```
Train Loss = 2.337, Train Accuracy = 0.081, Test Loss = 2.336, Test Accurac
y = 0.082
0.5885083315326008
0.7357198021507303
0.7627824796778018
0.7808364330889397
0.8249582356898796
0.7004838515997119
0.754607319887495
Train Loss = 0.599, Train Accuracy = 0.821, Test Loss = 0.757, Test Accurac
y = 0.790
```

In [ ]:
```
learning_rate = 5e-2

logs_8 = []

model = MyMLP()
print('Iteration 0', end=', ')
logs_8.append(compute_logs(model, verbose=True))

batch_size = 8

for j in range(50):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_8.append(log)
```

```
Iteration 0, Train Loss = 2.304, Train Accuracy = 0.130, Test Loss = 2.301,
Test Accuracy = 0.127
Iteration 1, Train Loss = 0.674, Train Accuracy = 0.766, Test Loss = 0.802,
Test Accuracy = 0.737
Iteration 2, Train Loss = 0.367, Train Accuracy = 0.873, Test Loss = 0.559,
Test Accuracy = 0.819
Iteration 3, Train Loss = 0.332, Train Accuracy = 0.875, Test Loss = 0.567,
Test Accuracy = 0.813
Iteration 4, Train Loss = 0.261, Train Accuracy = 0.906, Test Loss = 0.564,
Test Accuracy = 0.830
Iteration 5, Train Loss = 0.280, Train Accuracy = 0.904, Test Loss = 0.648,
Test Accuracy = 0.819
Iteration 6, Train Loss = 0.267, Train Accuracy = 0.902, Test Loss = 0.670,
Test Accuracy = 0.813
Iteration 7, Train Loss = 0.227, Train Accuracy = 0.919, Test Loss = 0.661,
Test Accuracy = 0.830
Iteration 8, Train Loss = 0.164, Train Accuracy = 0.942, Test Loss = 0.634,
Test Accuracy = 0.836
Iteration 9, Train Loss = 0.163, Train Accuracy = 0.940, Test Loss = 0.664,
Test Accuracy = 0.834
Iteration 10, Train Loss = 0.142, Train Accuracy = 0.947, Test Loss = 0.70
9, Test Accuracy = 0.832
Iteration 11, Train Loss = 0.181, Train Accuracy = 0.933, Test Loss = 0.77
1, Test Accuracy = 0.819
Iteration 12, Train Loss = 0.193, Train Accuracy = 0.929, Test Loss = 0.85
7, Test Accuracy = 0.814
Iteration 13, Train Loss = 0.123, Train Accuracy = 0.953, Test Loss = 0.80
4, Test Accuracy = 0.828
Iteration 14, Train Loss = 0.096, Train Accuracy = 0.965, Test Loss = 0.79
9, Test Accuracy = 0.836
Iteration 15, Train Loss = 0.130, Train Accuracy = 0.955, Test Loss = 0.90
0, Test Accuracy = 0.818
Iteration 16, Train Loss = 0.216, Train Accuracy = 0.938, Test Loss = 1.03
6, Test Accuracy = 0.810
Iteration 17, Train Loss = 0.086, Train Accuracy = 0.971, Test Loss = 0.89
9, Test Accuracy = 0.841
Iteration 18, Train Loss = 0.097, Train Accuracy = 0.963, Test Loss = 0.94
3, Test Accuracy = 0.825
Iteration 19, Train Loss = 0.062, Train Accuracy = 0.979, Test Loss = 0.93
7, Test Accuracy = 0.839
Iteration 20, Train Loss = 0.049, Train Accuracy = 0.982, Test Loss = 0.94
9, Test Accuracy = 0.837
Iteration 21, Train Loss = 0.046, Train Accuracy = 0.983, Test Loss = 0.95
6, Test Accuracy = 0.838
Iteration 22, Train Loss = 0.028, Train Accuracy = 0.991, Test Loss = 0.96
1, Test Accuracy = 0.836
Iteration 23, Train Loss = 0.051, Train Accuracy = 0.981, Test Loss = 1.04
5, Test Accuracy = 0.838
Iteration 24, Train Loss = 0.025, Train Accuracy = 0.993, Test Loss = 1.00
7, Test Accuracy = 0.835
Iteration 25, Train Loss = 0.061, Train Accuracy = 0.979, Test Loss = 1.12
3, Test Accuracy = 0.828
Iteration 26, Train Loss = 0.036, Train Accuracy = 0.987, Test Loss = 1.09
3, Test Accuracy = 0.836
Iteration 27, Train Loss = 0.018, Train Accuracy = 0.994, Test Loss = 1.08
3, Test Accuracy = 0.840
```

```
Iteration 28, Train Loss = 0.024, Train Accuracy = 0.993, Test Loss = 1.09
2, Test Accuracy = 0.838
Iteration 29, Train Loss = 0.023, Train Accuracy = 0.990, Test Loss = 1.15
6, Test Accuracy = 0.838
Iteration 30, Train Loss = 0.008, Train Accuracy = 0.998, Test Loss = 1.10
6, Test Accuracy = 0.841
Iteration 31, Train Loss = 0.018, Train Accuracy = 0.994, Test Loss = 1.17
8, Test Accuracy = 0.840
Iteration 32, Train Loss = 0.013, Train Accuracy = 0.995, Test Loss = 1.16
3, Test Accuracy = 0.842
Iteration 33, Train Loss = 0.006, Train Accuracy = 0.999, Test Loss = 1.13
2, Test Accuracy = 0.844
Iteration 34, Train Loss = 0.006, Train Accuracy = 0.999, Test Loss = 1.14
1, Test Accuracy = 0.845
Iteration 35, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.15
7, Test Accuracy = 0.846
Iteration 36, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.16
7, Test Accuracy = 0.844
Iteration 37, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.17
2, Test Accuracy = 0.845
Iteration 38, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.18
1, Test Accuracy = 0.845
Iteration 39, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.19
4, Test Accuracy = 0.844
Iteration 40, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.19
8, Test Accuracy = 0.843
Iteration 41, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.20
4, Test Accuracy = 0.843
Iteration 42, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.21
1, Test Accuracy = 0.844
Iteration 43, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.21
6, Test Accuracy = 0.844
Iteration 44, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.22
1, Test Accuracy = 0.843
Iteration 45, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.22
6, Test Accuracy = 0.843
Iteration 46, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.22
9, Test Accuracy = 0.844
Iteration 47, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.23
2, Test Accuracy = 0.843
Iteration 48, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.23
6, Test Accuracy = 0.844
Iteration 49, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.24
1, Test Accuracy = 0.844
Iteration 50, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.24
3, Test Accuracy = 0.844
```

3.6 Batchsize B=16

In [ ]:
```
learning_rate = 1.5e-1 #This is the divergent learning rate because loss dro
# We will use half of this learning rate to train the model

m = MyMLP()

_ = compute_logs(m, verbose=True)
```

```
m = minibatch_sgd_one_pass(m, X_train, y_train, learning_rate, batch_size=16

_ = compute_logs(m, verbose=True)
```

```
Train Loss = 2.358, Train Accuracy = 0.064, Test Loss = 2.363, Test Accurac
y = 0.066
0.538458054240026
0.635074907343773
0.6212223433196687
Train Loss = 0.523, Train Accuracy = 0.811, Test Loss = 0.631, Test Accurac
y = 0.777
```

In [ ]:
```
learning_rate = 1.5e-1/2

logs_16 = []

model = MyMLP()
print('Iteration 0', end=', ')
logs_16.append(compute_logs(model, verbose=True))

batch_size = 16

for j in range(50):
    model = minibatch_sgd_one_pass(model, X_train, y_train, learning_rate, b
    print(f'Iteration {j+1}', end=', ')
    log = compute_logs(model, verbose=True)
    logs_16.append(log)
```

```
Iteration 0, Train Loss = 2.321, Train Accuracy = 0.103, Test Loss = 2.319,
Test Accuracy = 0.103
Iteration 1, Train Loss = 0.420, Train Accuracy = 0.847, Test Loss = 0.524,
Test Accuracy = 0.817
Iteration 2, Train Loss = 0.365, Train Accuracy = 0.866, Test Loss = 0.540,
Test Accuracy = 0.815
Iteration 3, Train Loss = 0.301, Train Accuracy = 0.889, Test Loss = 0.501,
Test Accuracy = 0.830
Iteration 4, Train Loss = 0.295, Train Accuracy = 0.892, Test Loss = 0.549,
Test Accuracy = 0.824
Iteration 5, Train Loss = 0.248, Train Accuracy = 0.908, Test Loss = 0.545,
Test Accuracy = 0.833
Iteration 6, Train Loss = 0.215, Train Accuracy = 0.921, Test Loss = 0.564,
Test Accuracy = 0.831
Iteration 7, Train Loss = 0.189, Train Accuracy = 0.933, Test Loss = 0.548,
Test Accuracy = 0.835
Iteration 8, Train Loss = 0.230, Train Accuracy = 0.915, Test Loss = 0.627,
Test Accuracy = 0.818
Iteration 9, Train Loss = 0.267, Train Accuracy = 0.909, Test Loss = 0.718,
Test Accuracy = 0.806
Iteration 10, Train Loss = 0.140, Train Accuracy = 0.951, Test Loss = 0.59
9, Test Accuracy = 0.838
Iteration 11, Train Loss = 0.120, Train Accuracy = 0.960, Test Loss = 0.60
2, Test Accuracy = 0.841
Iteration 12, Train Loss = 0.104, Train Accuracy = 0.965, Test Loss = 0.62
0, Test Accuracy = 0.838
Iteration 13, Train Loss = 0.103, Train Accuracy = 0.962, Test Loss = 0.66
5, Test Accuracy = 0.833
Iteration 14, Train Loss = 0.104, Train Accuracy = 0.963, Test Loss = 0.71
4, Test Accuracy = 0.835
Iteration 15, Train Loss = 0.108, Train Accuracy = 0.964, Test Loss = 0.72
7, Test Accuracy = 0.833
Iteration 16, Train Loss = 0.069, Train Accuracy = 0.975, Test Loss = 0.73
0, Test Accuracy = 0.836
Iteration 17, Train Loss = 0.054, Train Accuracy = 0.982, Test Loss = 0.75
2, Test Accuracy = 0.846
Iteration 18, Train Loss = 0.061, Train Accuracy = 0.980, Test Loss = 0.77
8, Test Accuracy = 0.839
Iteration 19, Train Loss = 0.071, Train Accuracy = 0.973, Test Loss = 0.85
7, Test Accuracy = 0.833
Iteration 20, Train Loss = 0.084, Train Accuracy = 0.967, Test Loss = 0.85
6, Test Accuracy = 0.832
Iteration 21, Train Loss = 0.065, Train Accuracy = 0.980, Test Loss = 0.87
2, Test Accuracy = 0.828
Iteration 22, Train Loss = 0.147, Train Accuracy = 0.955, Test Loss = 1.00
8, Test Accuracy = 0.808
Iteration 23, Train Loss = 0.033, Train Accuracy = 0.990, Test Loss = 0.84
8, Test Accuracy = 0.840
Iteration 24, Train Loss = 0.055, Train Accuracy = 0.983, Test Loss = 0.93
0, Test Accuracy = 0.833
Iteration 25, Train Loss = 0.022, Train Accuracy = 0.994, Test Loss = 0.87
8, Test Accuracy = 0.843
Iteration 26, Train Loss = 0.022, Train Accuracy = 0.994, Test Loss = 0.92
2, Test Accuracy = 0.842
Iteration 27, Train Loss = 0.014, Train Accuracy = 0.996, Test Loss = 0.88
9, Test Accuracy = 0.841
```

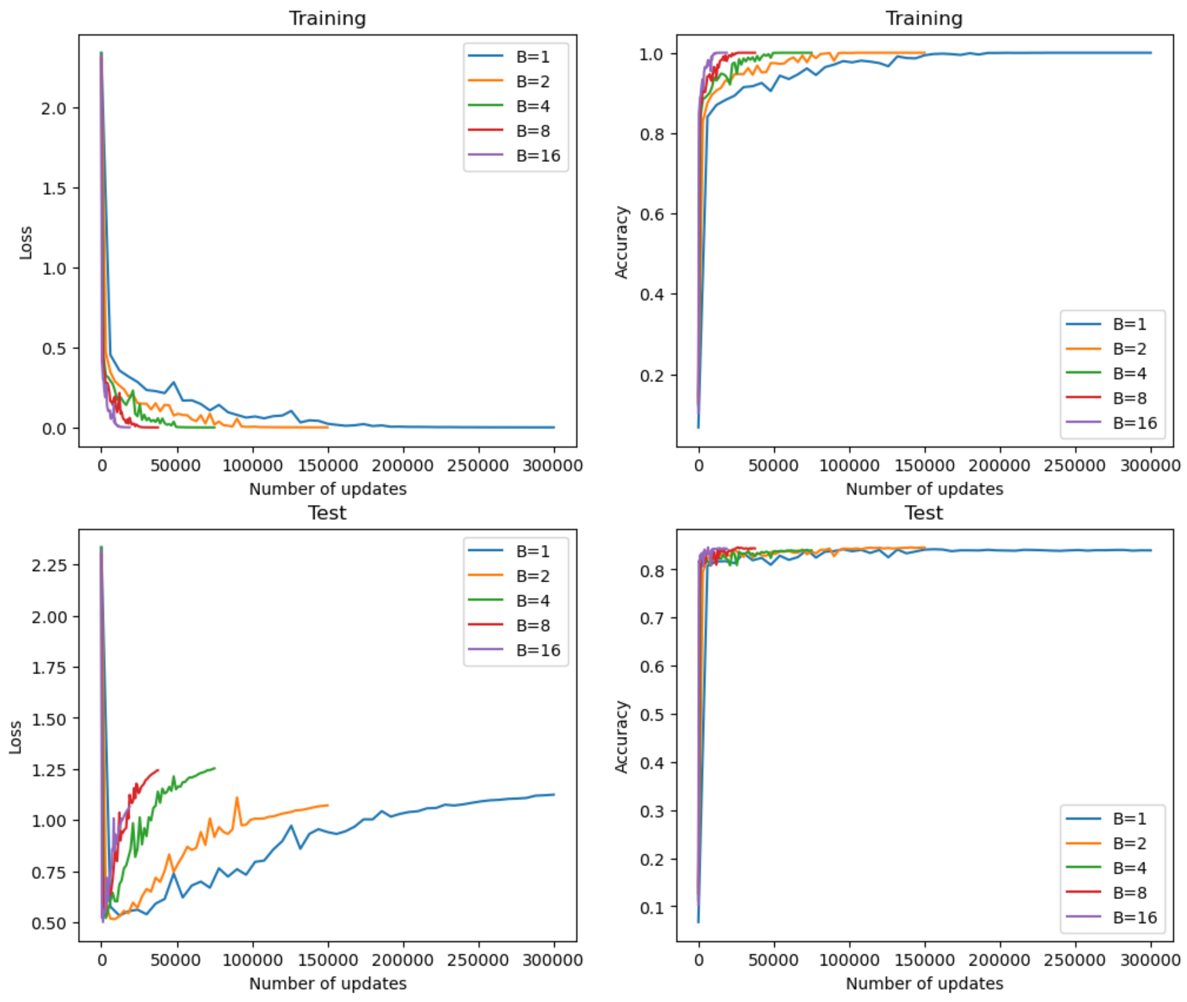
```
Iteration 28, Train Loss = 0.011, Train Accuracy = 0.998, Test Loss = 0.92
2, Test Accuracy = 0.843
Iteration 29, Train Loss = 0.009, Train Accuracy = 0.998, Test Loss = 0.92
5, Test Accuracy = 0.843
Iteration 30, Train Loss = 0.013, Train Accuracy = 0.997, Test Loss = 0.93
4, Test Accuracy = 0.842
Iteration 31, Train Loss = 0.007, Train Accuracy = 0.999, Test Loss = 0.96
4, Test Accuracy = 0.843
Iteration 32, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.95
6, Test Accuracy = 0.842
Iteration 33, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.97
1, Test Accuracy = 0.844
Iteration 34, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.98
0, Test Accuracy = 0.843
Iteration 35, Train Loss = 0.004, Train Accuracy = 1.000, Test Loss = 0.99
2, Test Accuracy = 0.843
Iteration 36, Train Loss = 0.005, Train Accuracy = 0.999, Test Loss = 1.00
7, Test Accuracy = 0.844
Iteration 37, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 0.99
9, Test Accuracy = 0.844
Iteration 38, Train Loss = 0.003, Train Accuracy = 1.000, Test Loss = 1.00
7, Test Accuracy = 0.843
Iteration 39, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.01
0, Test Accuracy = 0.843
Iteration 40, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.01
9, Test Accuracy = 0.843
Iteration 41, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.02
3, Test Accuracy = 0.842
Iteration 42, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
0, Test Accuracy = 0.842
Iteration 43, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
4, Test Accuracy = 0.843
Iteration 44, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.03
9, Test Accuracy = 0.844
Iteration 45, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.04
2, Test Accuracy = 0.843
Iteration 46, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.04
7, Test Accuracy = 0.843
Iteration 47, Train Loss = 0.002, Train Accuracy = 1.000, Test Loss = 1.05
2, Test Accuracy = 0.844
Iteration 48, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.05
6, Test Accuracy = 0.844
Iteration 49, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.06
0, Test Accuracy = 0.842
Iteration 50, Train Loss = 0.001, Train Accuracy = 1.000, Test Loss = 1.06
6, Test Accuracy = 0.843
```

3.7 Deliverables

Instead of obtaining the values from `compute_logs` at each update step we extrapolate the approximate number of update steps based on the batch size and the number of passes through the data.

In [ ]:
```
updates = [[i*X_train.shape[0]/batch for i in range(51)] for batch in [1,2,4
```

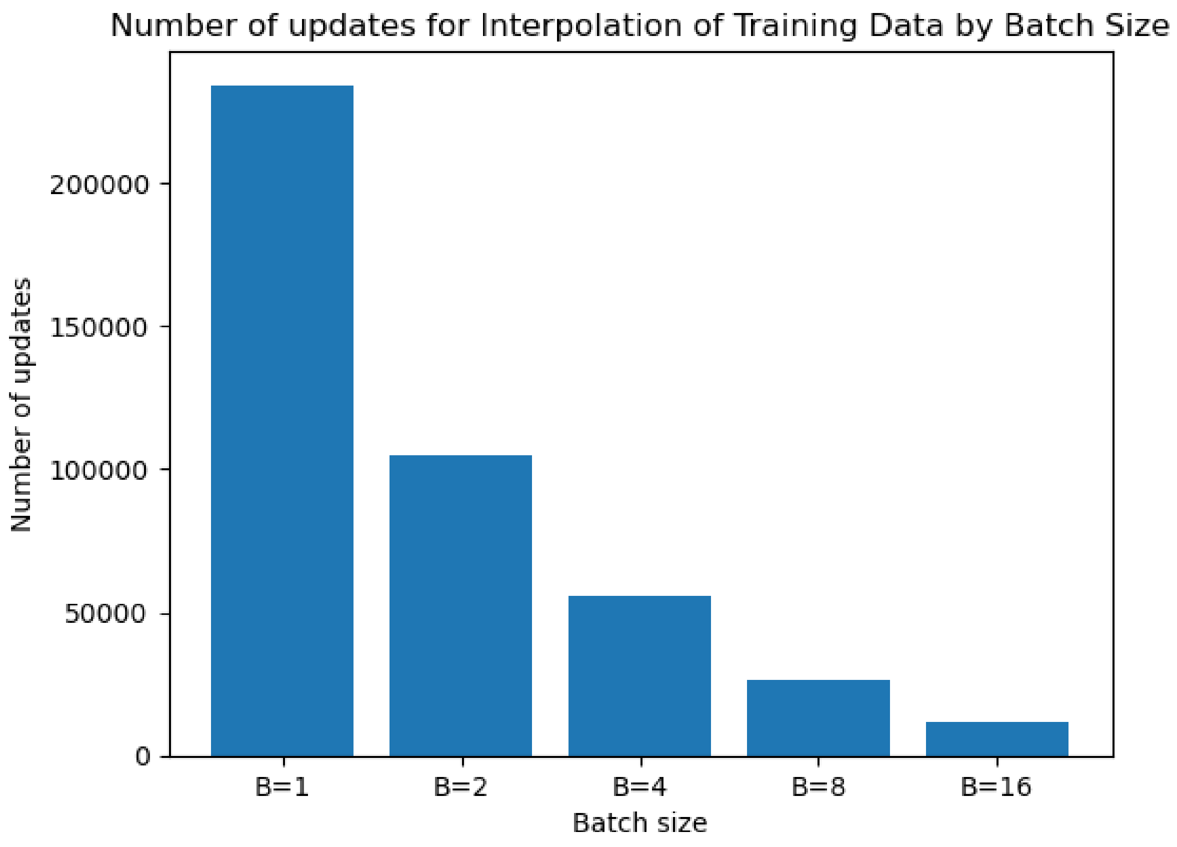
In [ ]:
```
fig, ax = plt.subplots(2,2)
fig.set_size_inches(12,10)
for a in ax.flatten(): a.set_xlabel("Number of updates")
for a in ax: a[0].set_ylabel("Loss")
for a in ax: a[1].set_ylabel("Accuracy")
for a in ax[0,:]: a.set_title("Training")
for a in ax[1,:]: a.set_title("Test")

names = zip([logs_1, logs_2, logs_4, logs_8, logs_16], ["B=1", "B=2", "B=4",
for i,(log, label) in enumerate(names):
    log = np.asarray(log)
    ax[0,0].plot(updates[i], log[:,0], label = label)
    ax[0,0].legend()
    ax[0,1].plot(updates[i], log[:,1], label = label)
    ax[0,1].legend()
    ax[1,0].plot(updates[i], log[:,2], label = label)
    ax[1,0].legend()
    ax[1,1].plot(updates[i], log[:,3], label = label)
    ax[1,1].legend()
```



In [ ]:
```
def find_interpolation_index(log):
    log = np.asarray(log)
    idx = np.where(log[:,1]==1.0)[0]
    return idx.min()

idx = {}
names = zip([logs_1, logs_2, logs_4, logs_8, logs_16], ["B=1", "B=2", "B=4",
for i, (log, label) in enumerate(names):
    x = find_interpolation_index(log)
    step = updates[i][1] - updates[i][0]
    idx[label] = x*step
print(idx)
```

```
{'B=1': 234000.0, 'B=2': 105000.0, 'B=4': 55500.0, 'B=8': 26250.0, 'B=16':
12000.0}
```

In [ ]:
```
plt.bar(idx.keys(), idx.values());
plt.xlabel("Batch size");
plt.ylabel("Number of updates")
plt.title("Number of updates for Interpolation of Training Data by Batch Siz
```



4. (Bonus) Serial updates and gradient computations with minibatching (continued)

In this exercise, we will explore the scenario where computing a minibatch stochastic gradient is roughly B times as expensive as computing a stochastic gradient with a batch size of 1.

Here are the details:

- The dataset, model and learning rate as same as in Exercise 3.

- Train the model for 50 passes through the data. If n is the number of training examples, this corre- sponds to 50n/B passes over the data with a batch size of B.

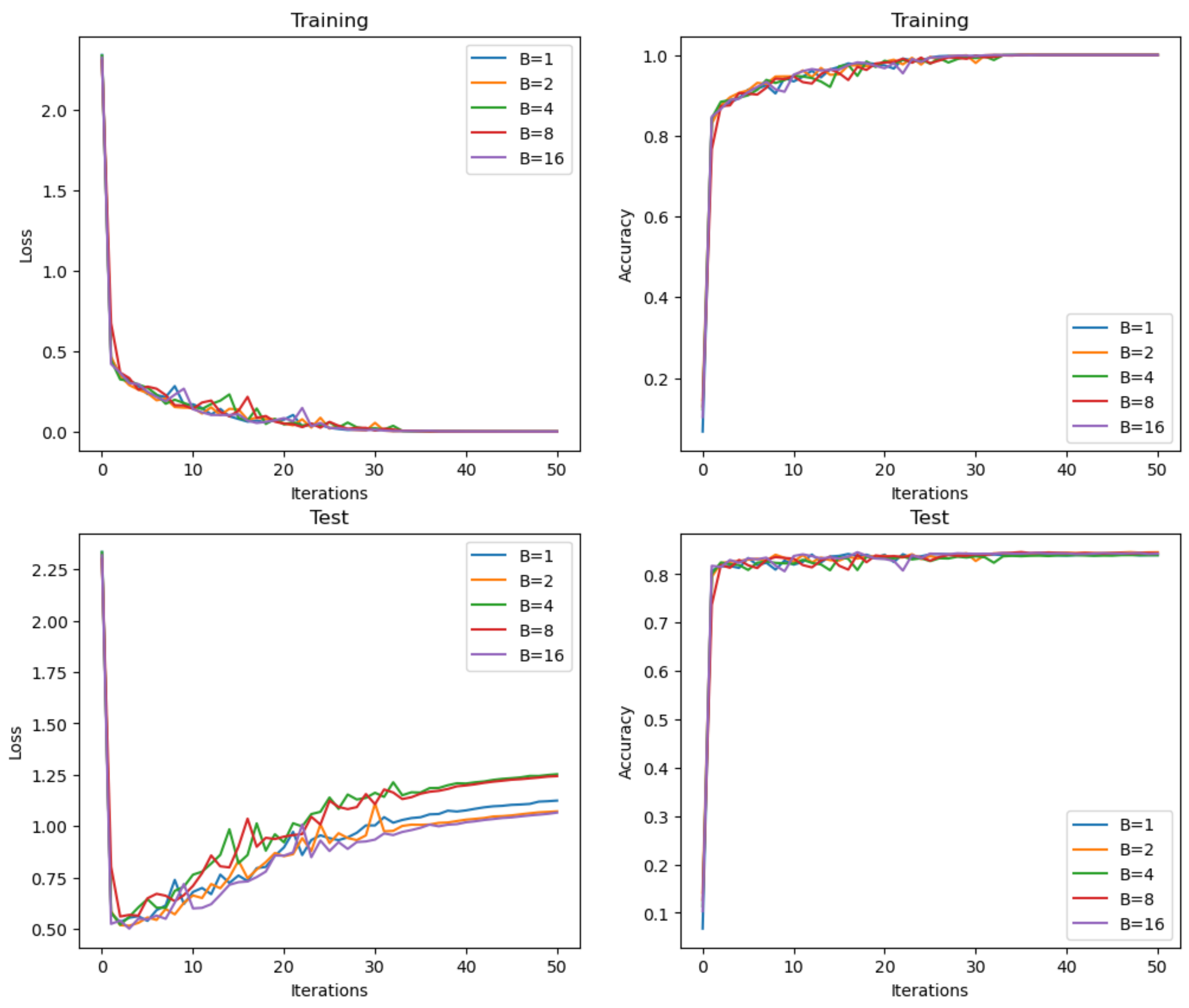
The deliverables for this exercise are:

1. Make 4 plots, one each for the train loss, train accuracy, test loss and test accuracy over the course of training (i.e., the metric on the y-axis and *number of effective passes* on the x-axis). Plot all 5 lines, one for each value of B on the same plot.
2. As we vary the batch size of the network, at which training epoch do we observe perfect interpolation of the data? That is, make a plot with B on the x-axis and number of passes over the data required for interpolation on the y axis.

Deliverables

This part is simpler, we simply plot the values obtained from `compute_logs` against the number of passes through the training data.

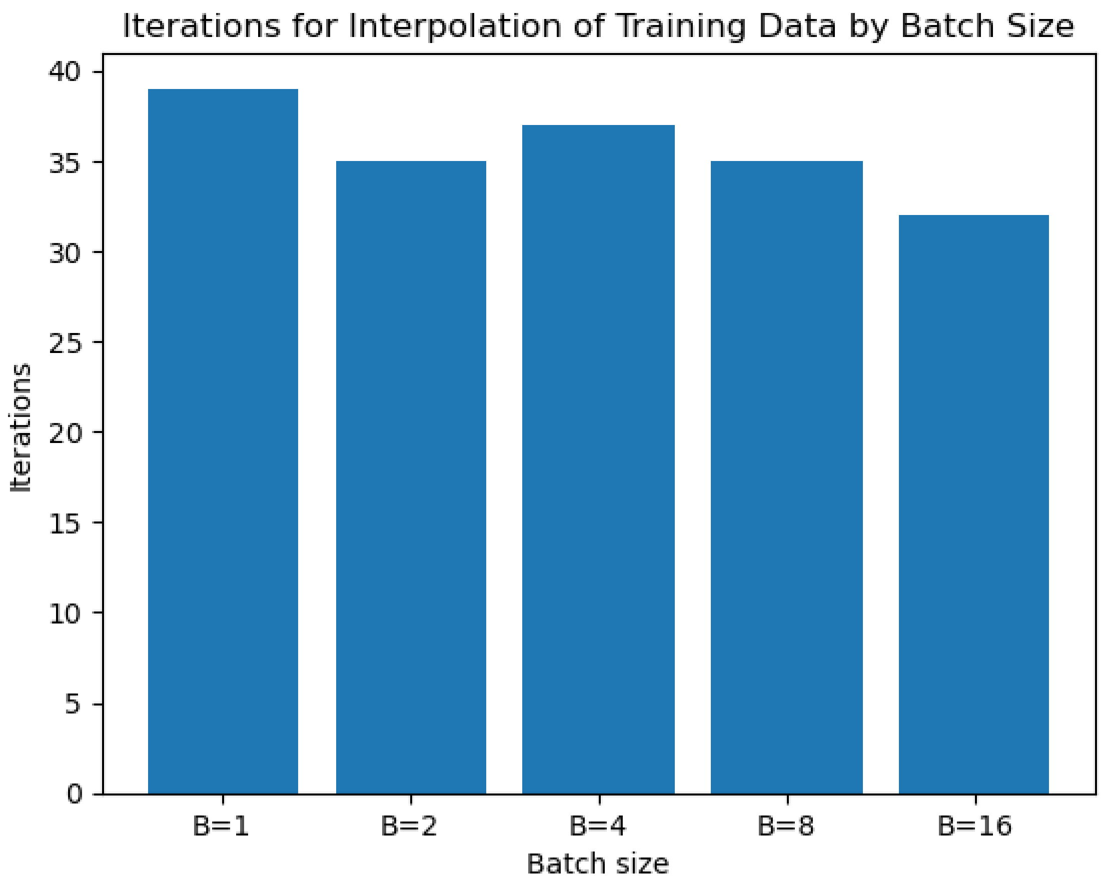
In [ ]:
```python
fig, ax = plt.subplots(2,2)
fig.set_size_inches(12,10)
for a in ax.flatten(): a.set_xlabel("Iterations")
for a in ax: a[0].set_ylabel("Loss")
for a in ax: a[1].set_ylabel("Accuracy")
for a in ax[0,:]: a.set_title("Training")
for a in ax[1,:]: a.set_title("Test")

names = zip([logs_1, logs_2, logs_4, logs_8, logs_16], ["B=1", "B=2", "B=4",
for log, label in names:
    log = np.asarray(log)
    ax[0,0].plot(log[:,0], label = label)
    ax[0,0].legend()
    ax[0,1].plot(log[:,1], label = label)
    ax[0,1].legend()
    ax[1,0].plot(log[:,2], label = label)
    ax[1,0].legend()
    ax[1,1].plot(log[:,3], label = label)
    ax[1,1].legend()
```



```
In [ ]: def find_interpolation_index(log):
            log = np.asarray(log)
            idx = np.where(log[:,1]==1.0)[0]
            return idx.min()

        idx = {}
        names = zip([logs_1, logs_2, logs_4, logs_8, logs_16], ["B=1", "B=2", "B=4",
        for log, label in names:
            idx[label] = find_interpolation_index(log)
        print(idx)
```

```
{'B=1': 39, 'B=2': 35, 'B=4': 37, 'B=8': 35, 'B=16': 32}
```

```
In [ ]: plt.bar(idx.keys(), idx.values());
        plt.xlabel("Batch size");
        plt.ylabel("Iterations")
        plt.title("Iterations for Interpolation of Training Data by Batch Size");
```


Iterations for Interpolation of Training Data by Batch Size
Iterations
0
5
10
15
20
25
30
35
40
B=1
B=2
B=4
B=8
B=16
Batch size