In [1]:
```
import numpy as np
import torch

from torch.utils.data import DataLoader
from torch import optim

from torchvision.datasets import MNIST
from torchvision import transforms as T


import utils
import solution
```

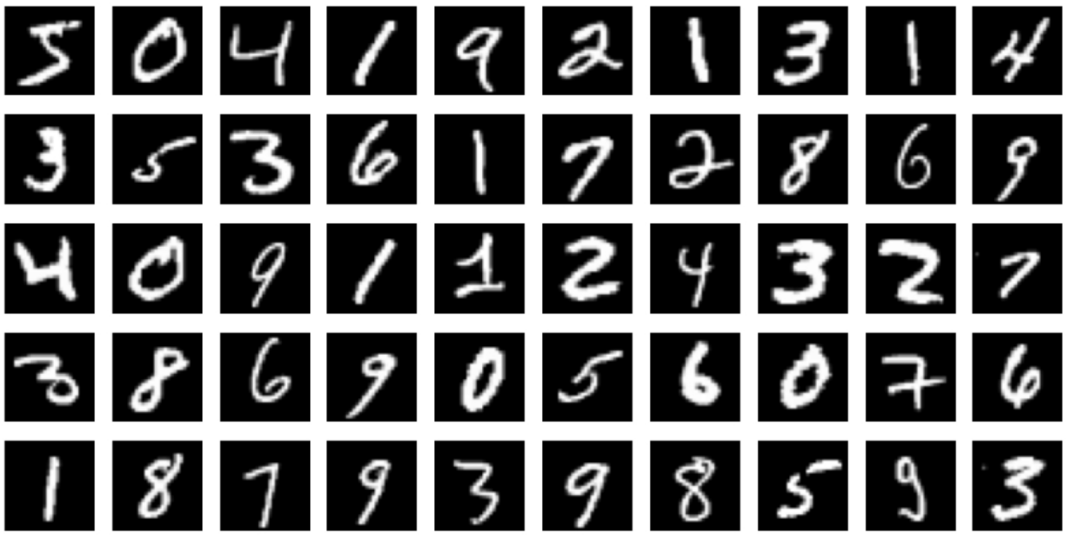
1 Introduction

Momentum has been introduced in the lecture as a mechanism to speed up training. We would like to show this empirically in this exercise by an example on the MNIST dataset. We use a simple network architecture to classify handwritten digits into 10 different classes (0-9).

Your task is to implement some parts of the training loop and study the effect of momentum with respect to the training speed. We compare training the model with Stochastic Gradient Descent (SGD) without momentum and with a momentum of `0.9`. To highlight that the effect is not the result of a specific hyperparameter choice, we train with different batch sizes and learning rates. For each of these configurations, we train two models: one with SGD and the other with SGD and momentum.

In [2]:
```
# loading data
data_root = './data'
train_dataset = MNIST(data_root, train=True, download=True, transform=T.ToTensor())
test_dataset = MNIST(data_root, train=False, download=True, transform=T.ToTensor())
```

In [3]:
```
# visualizing samples
utils.show_samples(train_dataset)
```



2 Training Function Implementation (20 P)

Task: Implement the training function based on the predefined `train_one_epoch` function provided in `utils.py`. The training function receives a random initialized model and various hyperparameters. Your task is to implement the training loop given the function `train_one_epoch`. We want to optimize the model with a cross entropy loss. Further, we want to collect the final train and test accuracy and also how the training metrics progress over the training.

Therefore, also return the train loss and accuracy after each epoch of the training for further analysis.

Hint: to compute the final training and test accuracy after training, you can use the `accuracy` function from `utils.py`

In [4]:
```
def train(model, train_dataset, test_dataset, epochs=10, batch_size=32, lr=0.01, momentum=0.0
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size)
    optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)

    # YOUR CODE HERE

    return train_acc, test_acc, arr_epoch_loss, arr_epoch_train_accuracy

train = solution.train
```

In [5]:
```
# sanity check for model training
# with 1 epoch, the test accuracy should be ~80%.
torch.manual_seed(1)
model = utils.Lenet5()
train(model=model, train_dataset=train_dataset, test_dataset=test_dataset, epochs=1);
```

```
Train Accuracy: 0.8053; Test Accuracy: 0.8097
```

In [6]:
```
# with 1 epoch, the test accuracy should be ~96%.
torch.manual_seed(1)
model2 = utils.Lenet5()
train(model=model2, train_dataset=train_dataset, test_dataset=test_dataset, epochs=1, momentu
```

```
Train Accuracy: 0.9667; Test Accuracy: 0.9683
```

3 Visualizing the effect of momentum with varying the value of learning rate (15 P)

Task: We want to compare the effect of momentum on the optimization process for different learning rates. Use the previously created training function to implement the function `sweep_lr` which should train a model without momentum (`momentum=0.0`) and momentum (`momentum=0.9`). For each of the two trained models, plot the loss value after each epoch to study how the loss progresses during training.

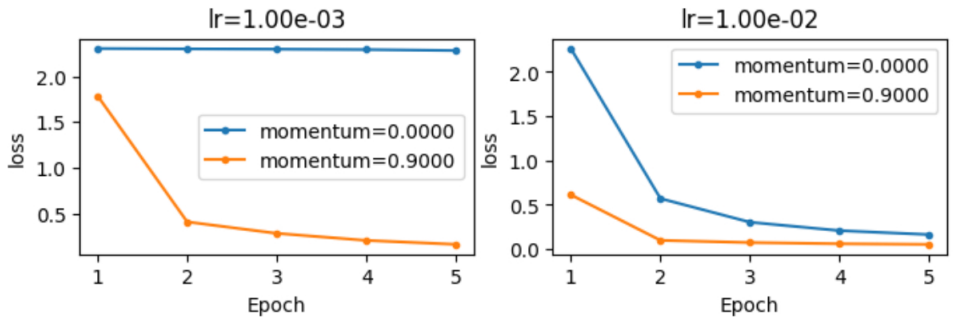
What can you observe from these plots? What influence does momentum have on the training and why?

In [7]:
```
def sweep_lr(arr_lr, train_dataset, test_dataset, epochs=5):
    torch.manual_seed(1)

    # YOUR CODE HERE

sweep_lr = solution.sweep_lr
sweep_lr([1e-3, 1e-2], train_dataset, test_dataset)
```

```
Train Accuracy: 0.2090; Test Accuracy: 0.2106
Train Accuracy: 0.9599; Test Accuracy: 0.9625
Train Accuracy: 0.9574; Test Accuracy: 0.9603
Train Accuracy: 0.9855; Test Accuracy: 0.9830
```



4 Visualizing the effect of momentum when varying the value of batch size (15 P)

Task: We want to compare the effect of momentum on the optimization process for different batch sizes. Similar to the `sweep_lr` function, implement the function `sweep_batchsize` which should train a model without momentum (`momentum=0.0`) and momentum (`momentum=0.9`) with the specified batch size. For each of the two trained models, plot the loss value after each epoch to study how the loss progresses during training.

What can you observe from these plots?

In [8]:
```
def sweep_batchsize(arr_batchsize, train_dataset, test_dataset, epochs=5):
    torch.manual_seed(1)
```

```
    # YOUR CODE HERE

sweep_batchsize = solution.sweep_batchsize

sweep_batchsize([16, 32, 64], train_dataset, test_dataset)
```

```
Train Accuracy: 0.9740; Test Accuracy: 0.9724
Train Accuracy: 0.9858; Test Accuracy: 0.9851
Train Accuracy: 0.9574; Test Accuracy: 0.9603
Train Accuracy: 0.9855; Test Accuracy: 0.9830
Train Accuracy: 0.9277; Test Accuracy: 0.9297
Train Accuracy: 0.9847; Test Accuracy: 0.9851
```