KAIST

AI502 Deep Learning

Homework 1 - simple Regression and Classification problem

Cappa Victor
20206080
School of computing
victor.cappa@kaist.ac.kr

HW1-1 - Polynomial approximation using Pytorch

In the given section we work on the provided “[HW1-1] Polynomial Approximation.ipynb” jupyter notebook. The task we want to solve is to approximate a target polynomial function f(x) given a randomly generated dataset of data points <x, f(x)> and train the neural network in order to approximate the coefficients of the target polynomial function.

The adopted parameters are:

- Smooth function: SmoothL1Loss
- Optimizer: SGD without momentum
- Number of epochs: 500
- Input dataset size: 200
- Batch size: 50
- Learning rate: 0.1, 0.01, 0.001
- No learning rate scheduler
- No division between train, validation and test sets

We want to explore how the approximation of the target function (the parameters learnt by the ANN) evolves during time and the different behaviours that can be achieved by using different learning rates.
In order to do this two different graphs are plotted:

- plot of the real function and the learnt function (done for epoch 5, 10, 20, 100, 200, 500)
- plot of the loss (computed for each epoch and plotted on the graph)

The first graph will be useful to understand how the learnt function is converging to the target function in the different epochs and the second one will help us to analyze the convergence of the neural network to the final solution.

The seed 2020 has been adopted for convenience.
In order to plot the loss, the target function and the learnt functions the following python code have been implemented or modified:

```
def compute_results(x, w, bias):
        x = make_features(x)
        return torch.squeeze(x.mm(w) + bias[0])
```

Function for plotting the Actual-Learned function plot:

```
def plot_graphs(W_target, b_target, W_learned, b_learned, epoch_num):
        x = torch.linspace(-10, 10, steps=1000)
        fig = plt.figure(figsize=(10, 5))
        plt.title("Function plot - epoch number: {}".format(epoch_num))
        plt.xlabel("x-axis")
        plt.ylabel("y-axis")
        plt.grid()
        plt.plot(x.numpy(), compute_results(torch.linspace(-10, 10, steps=1000), W_target,
b_target).numpy(), label="Actual function")
        plt.plot(x.numpy(), compute_results(x, torch.unsqueeze(torch.squeeze(W_learned), 1),
b_learned), label="Learned function")
        plt.legend()
        plt.show()
```

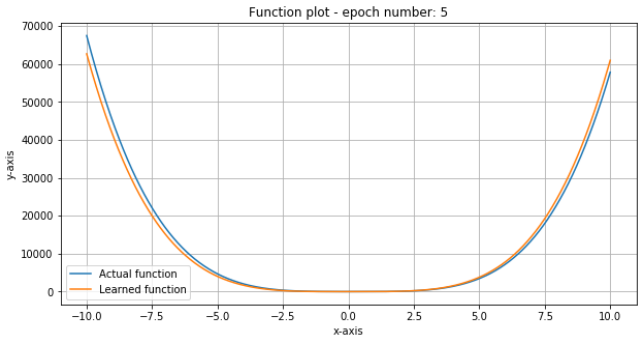
Modified fit function:

```
def fit(model,loader,criterion,learning_rate,num_epochs, graphs=False):
        model.train()
        optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
        loss_list = []
        for epoch in range(num_epochs):
        accum_loss = 0
        num_batch = 0
        for i, data in enumerate(loader):
        if torch.cuda.is_available():
                x = data[0].type(torch.FloatTensor).cuda()
                y = data[1].type(torch.FloatTensor).cuda()
        else:
                x = data[0].type(torch.FloatTensor)
                y = data[1].type(torch.FloatTensor)
        y_hat = model(x)
        loss = criterion(y_hat, y)
        accum_loss += loss.item()
        num_batch += 1
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_list.append(accum_loss/num_batch)
        if graphs==True and (epoch+1) in [5,10,20,100,200, 500]:
                plot_graphs(W_target, b_target, model.fc.weight.data, model.fc.bias.data, epoch+1)
                print('==> Learned function:\t' + poly_desc(model.fc.weight.data.view(-1),
model.fc.bias.data))
                print('==> Actual function:\t' + poly_desc(W_target.view(-1), b_target))
        return loss_list
```

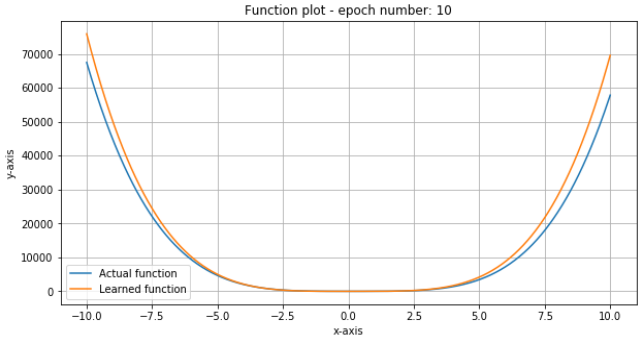
Code for plotting the loss function:

```
fig = plt.figure(figsize=(10, 5))
plt.title("Loss function - Learning rate: {}".format(learning_rate))
plt.xlabel("epoch")
plt.ylabel("loss")
plt.grid()
plt.plot(list(range(num_epochs)), loss_list)
plt.show()
```

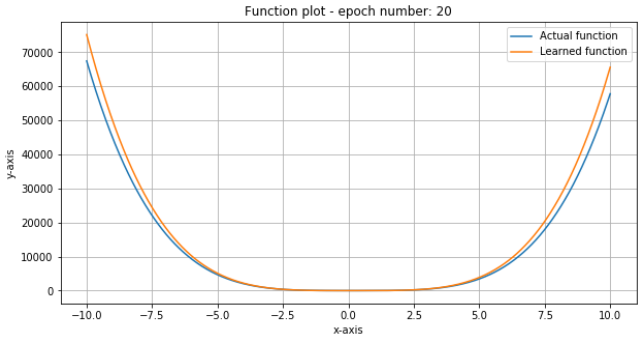
1) Plots with learning rate = 0.1



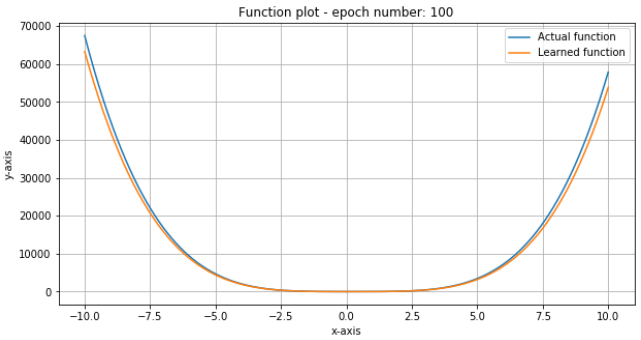
==> Learned function: y = +6.16 x^4 -0.86 x^3 +1.87 x^2 -0.14 x^1 +1.95
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



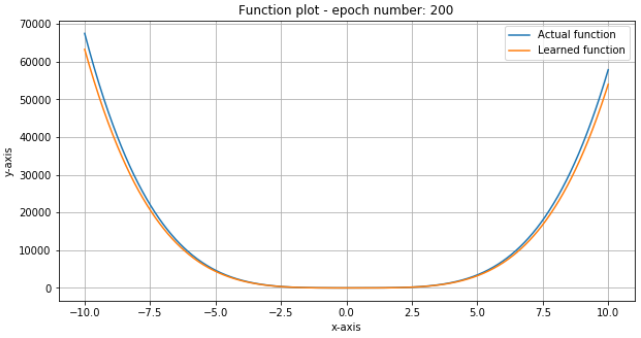
==> Learned function: y = +7.25 x^4 -3.16 x^3 +2.56 x^2 -0.83 x^1 +3.54
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



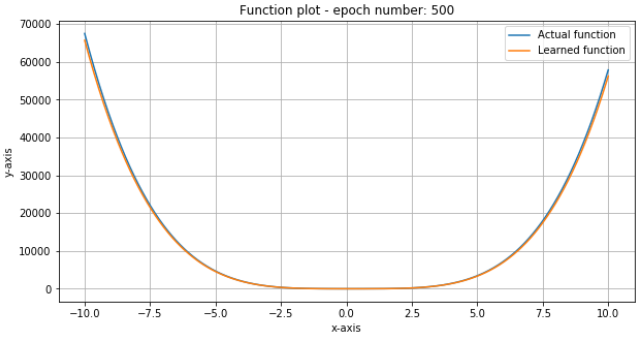
==> Learned function: y = +7.00 x^4 -4.77 x^3 +4.30 x^2 -1.57 x^1 +5.33
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



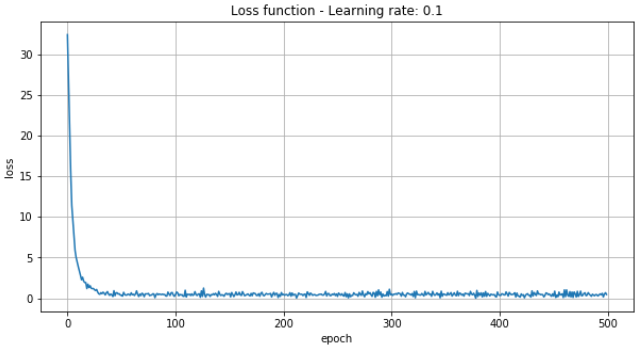
==> Learned function: y = +5.77 x^4 -4.71 x^3 +7.73 x^2 -2.11 x^1 +4.46
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.4



==> Learned function: y = +5.78 x^4 -4.61 x^3 +7.62 x^2 -1.98 x^1 +4.39
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40

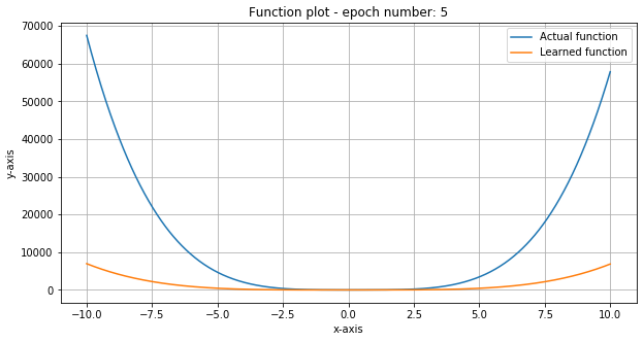


==> Learned function: y = +6.01 x^4 -4.72 x^3 +7.70 x^2 -2.03 x^1 +4.46
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40

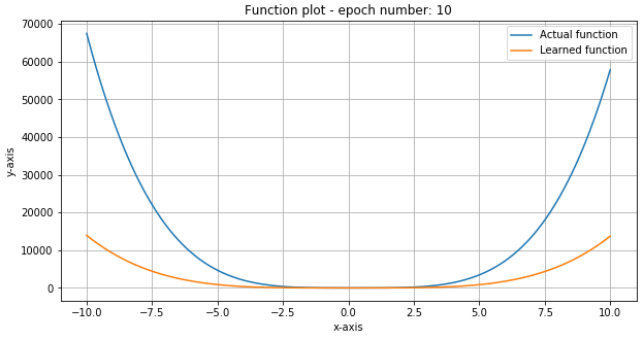


Loss function plot

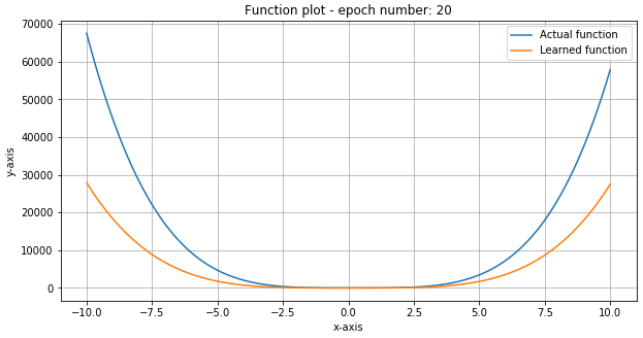
2) Plots with learning rate = 0.01



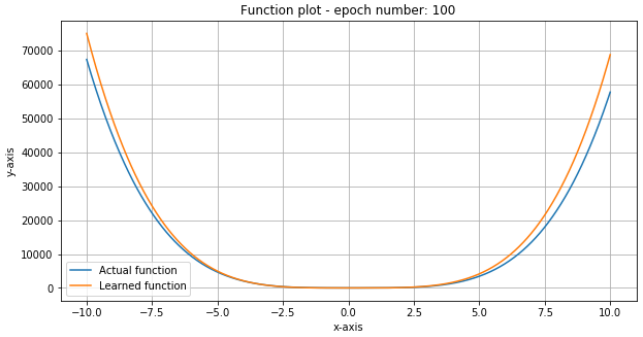
==> Learned function: y = +0.69 x^4 -0.05 x^3 +0.20 x^2 -0.00 x^1 +0.20
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



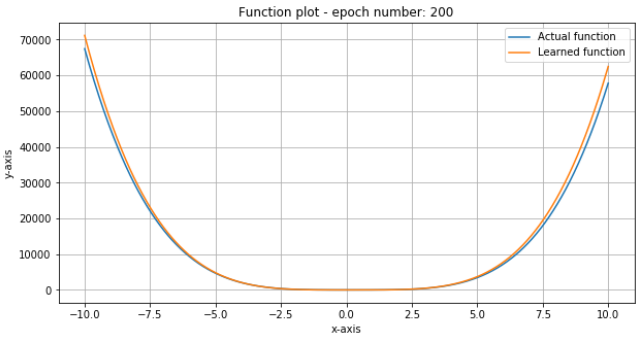
==> Learned function: y = +1.38 x^4 -0.10 x^3 +0.41 x^2 -0.01 x^1 +0.40
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



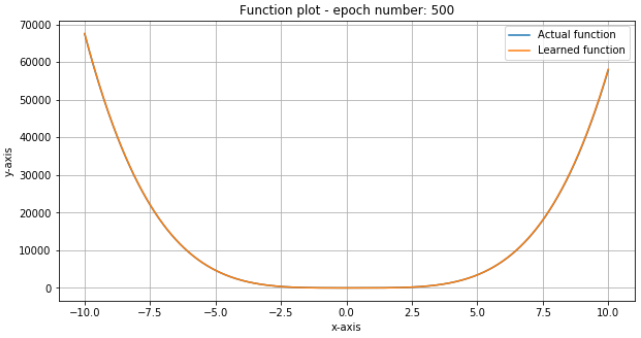
==> Learned function: y = +2.76 x^4 -0.20 x^3 +0.82 x^2 -0.02 x^1 +0.80
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



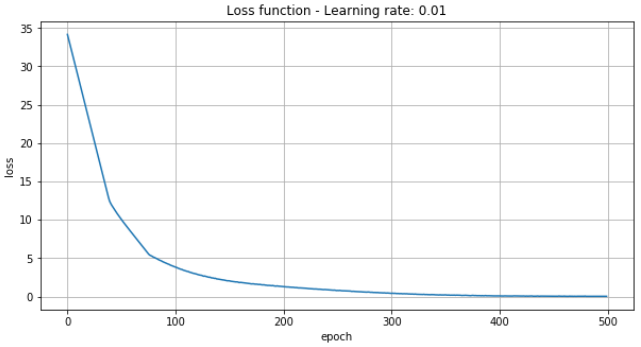
==> Learned function: y = +7.17 x^4 -3.13 x^3 +2.57 x^2 -0.82 x^1 +3.54
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



==> Learned function: y = +6.64 x^4 -4.34 x^3 +4.45 x^2 -1.49 x^1 +5.34
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40

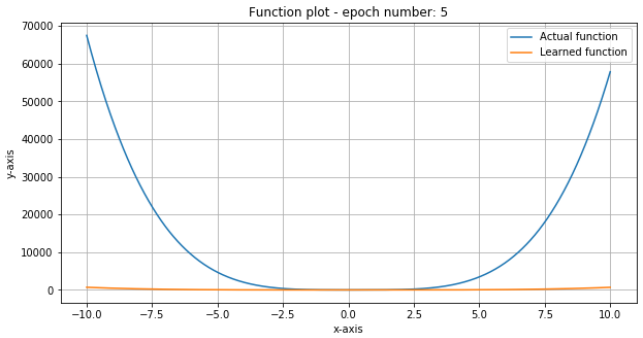


==> Learned function: y = +6.21 x^4 -4.75 x^3 +7.51 x^2 -2.16 x^1 +4.55
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40

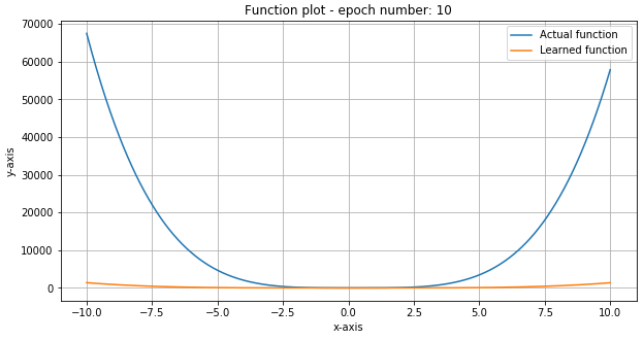


Loss function plot

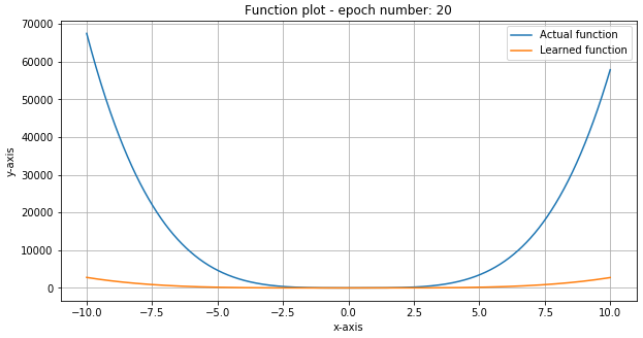
3) Plots with learning rate = 0.001



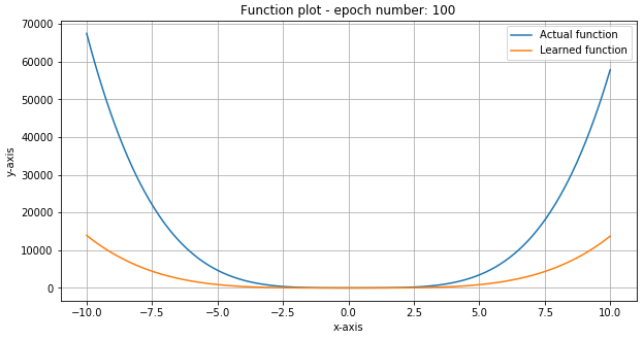
==> Learned function: y = +0.07 x^4 -0.00 x^3 +0.02 x^2 -0.00 x^1 +0.02
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



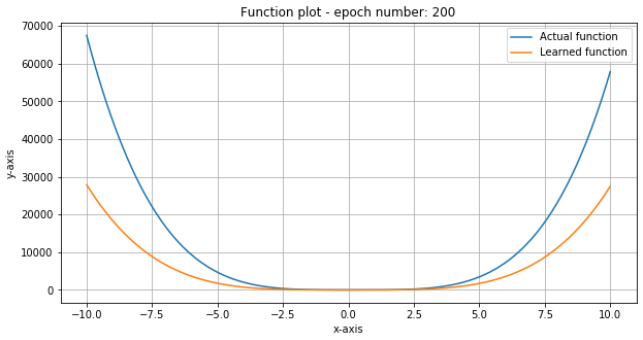
==> Learned function: y = +0.14 x^4 -0.01 x^3 +0.04 x^2 -0.00 x^1 +0.04
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



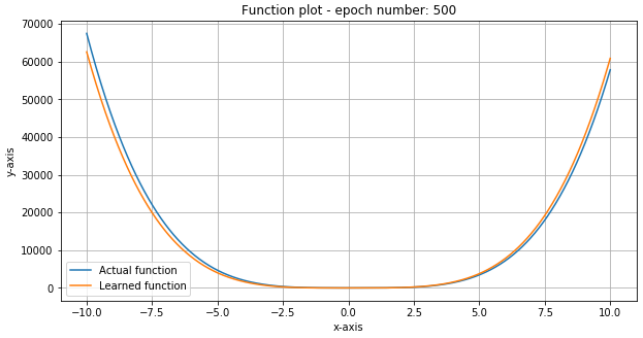
==> Learned function: y = +0.28 x^4 -0.02 x^3 +0.08 x^2 -0.00 x^1 +0.08
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



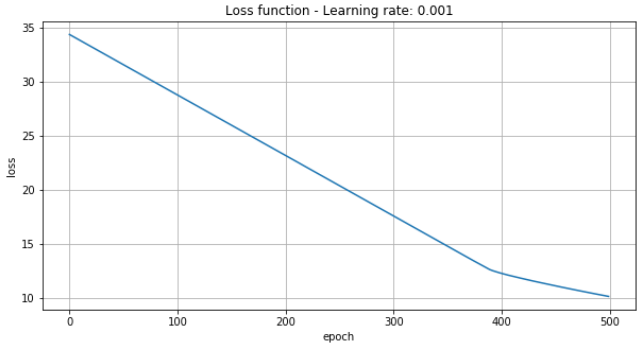
==> Learned function: y = +1.38 x^4 -0.10 x^3 +0.41 x^2 -0.01 x^1 +0.40
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



==> Learned function: y = +2.76 x^4 -0.20 x^3 +0.82 x^2 -0.02 x^1 +0.80
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



==> Learned function: y = +6.15 x^4 -0.85 x^3 +1.87 x^2 -0.14 x^1 +1.95
==> Actual function: y = +6.19 x^4 -4.80 x^3 +7.71 x^2 -2.04 x^1 +4.40



Loss function plot

Plotting the target function and the learned functions for different stages of the learning process and plotting the loss function is very interesting in order to understand the behaviour of the training phase of the neural network for different values of the learning rate.
It is possible to conclude that choosing a learning rate of 0.1 allows the neural network to quickly converge to a final solution in a very small number of epochs. This is confirmed by the loss function plot where we can see that the minimum value of the loss is already achievable after around 20 epochs and after this time the neural network isn't improving anymore the loss function value, meaning that a learning rate scheduler could probably be a proper solution.
On the other hand by choosing a learning rate of 0.01 the neural network converges much slowly to a final solution, it converges in approximately 400 epochs, but the resulting final loss is much lower

compared to the previous one (found with learning rate of 0.1) and the resulting quality of the learned function is also higher as we can see from the target function-learned function plot. Moreover, from this plot we can also see how the learned function gradually converges over time to the target function.
Finally, a learning rate of 0.001 was chosen. As we can see after 500 epochs the loss function is still monotonically decreasing, meaning the learning rate is too small and with the given number of epochs and the given configuration the neural network is not able to converge to a final solution.
To conclude, the best solution with the given configuration would be choosing a initial learning rate of 0.1 with a properly tuned learning rate scheduler, diminishing the learning rate after a fixed number of epochs.

HW1-2 - Classification task using MNIST data

[Question 1] What is 'torch.cuda.is_available' used for?
torch.cuda.is_available is used in order to determine whether the host machine supports cuda or not. CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia which allows to use a CUDA-enabled graphics processing unit (GPU), allowing to speed up the training of neural networks especially when they require a lot of computation.

[Question 2] What is 'torch.utils.data.DataLoader' used for?
The DataLoader class in pytorch permits to combine the input dataset with a sampler, and provides an iterable over the given dataset. It allows to return batches of data of a fixed size and also has other useful functionalities for data retrieval like batch_size, shuffle, num_workers, drop_last, pin_memory and so on.

[Question 3] What is 'nn.Linear' used for?
nn.Linerar is a pytorch class allowing to apply a linear transformation (matrix of parameters and bias vector) of an input tensor. The returned object is another tensor.
A code example could be the following:
m = nn.Linear(20,30)
input_tensor = torch.randn(128, 20)
output = m(input_tensor)

[Question 4] (4-1)What is 'view' used for? (4-2)What does '-1' mean?
view is a Tensor class' method used to return a new tensor with the same data as the self tensor but of a different shape, specified in the method's arguments. The returned tensor shares the same data and must have the same number of elements.
When -1 is used it means that the given size must be inferred from the other dimensions.

[Question 5] What is 'optim.SGD' used for?
SGD is a class that can be found in the optim package of pytorch. This package contains all the available optimization algorithms and SGD implements the Stochastic gradient descent algorithm, also with momentum. It is used to update the model's parameters and accepts as an argument an iterable of the parameters to optimize, the learning rate, momentum and others.

[Question 6] What is 'nn.CrossEntropyLoss' used for?
The class CrossEntropyLoss is used for computing the loss of a given output of the neural network, with respect to the ground truth. It is the loss function that we use to train the model.

[Question 7] What is 'optimizer.zero_grad()' used for?
It clears all the previously computed gradients.

[Question 8] What are 'out' and 'target' and what is output of criterion(out, target)?
The input of criterion() is expected to contain raw, unnormalized scores for each class. out is the output tensor of the neural network, while target is a one hot encoded vector (usually) corresponding to the ground truth vector. The output of it is a one dimensional pytorch tensor containing the loss. Starting from the given loss I can compute the gradients by executing the backward() method.

[Question 9] What is 'loss.backward()' used for?
loss.backward() is used to compute the gradients for each trainable parameter, starting from the returned loss.

[Question 10] What is 'optimizer.step()' used for?
It is used to update the model's parameters based on the previously computed gradients. It performs a single optimization step.