Exercise Solution 04

Exercise Solution 04

MLP

MLP Classifier for images

We define the new MLPClassifier class, mostly based on the previous MLP class that was given. We only add a flattening layer at the beginning of the network. This is done by adding the line

```
layers.append(nn.Flatten(start_dim=1, end_dim=-1))
```

to the `__init__` method of the MLP class. The rest of the code is the same as the previous MLP class.

Fitting

To fit the model we first need to initialize it. We create the MLP with an input layer of 784 nodes, followed by two hiden layers of 40 nodes, and an output layer of 10 nodes. We use the `torch.optim.Adam` optimizer and the `torch.nn.CrossEntropyLoss` loss function.

```
nunits = [784, 40, 40, 10]
model = MLPClassifier(nunits)
optimizer = torch.optim.Adam(model.parameters())
loss = nn.CrossEntropyLoss()
```

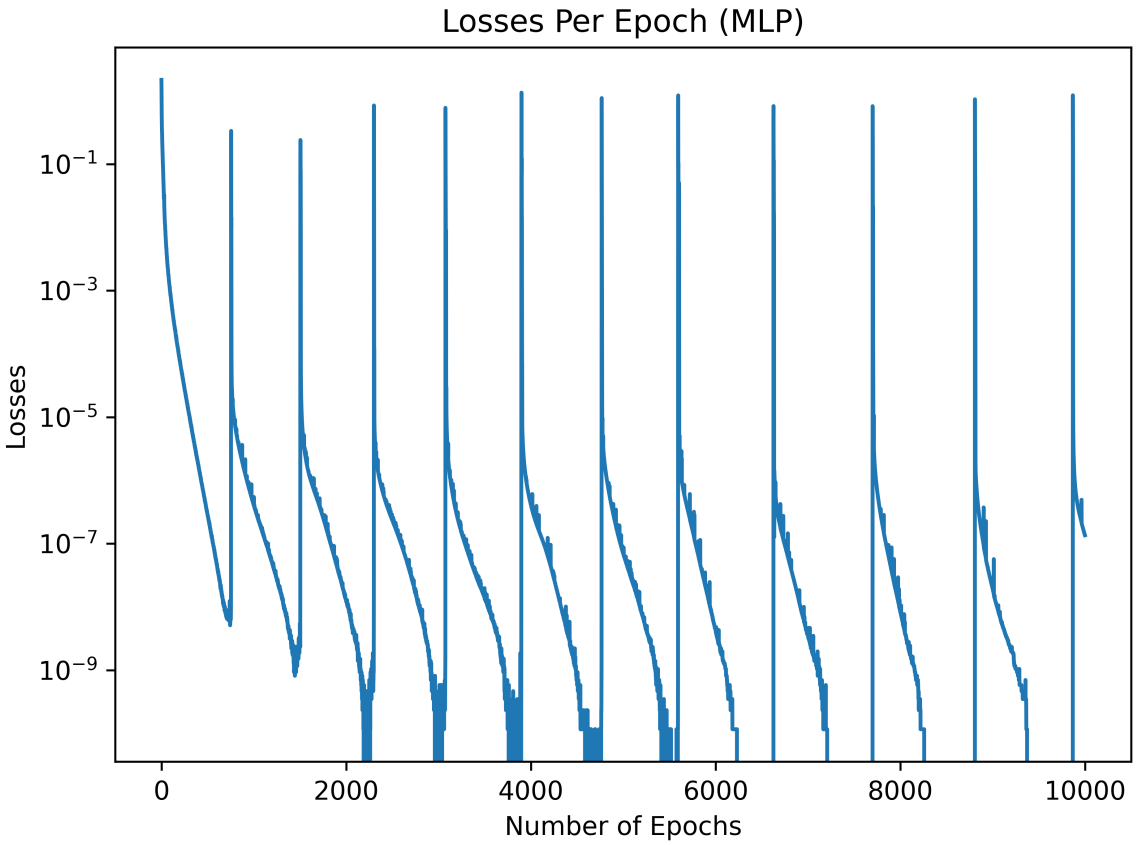
The training loop is the same as the one used in the previous exercises. We train the model for 10000 epochs.

```
nepochs = 10000
losses= np.zeros(nepochs)
classification_accuracies = np.zeros(nepochs)

for epoch in range(nepochs):
    losses[epoch] = train(model, train_loader, optimizer, loss)
    _, accuracy = evaluate_classification(model, loss, test_loader)
    classification_accuracies[epoch] = accuracy.item()
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} || Losses: {losses[epoch]}")
save_model(model, "mlp.pt")
np.save("classification_accuracies.npy", classification_accuracies)
np.save("losses.npy", losses)
```

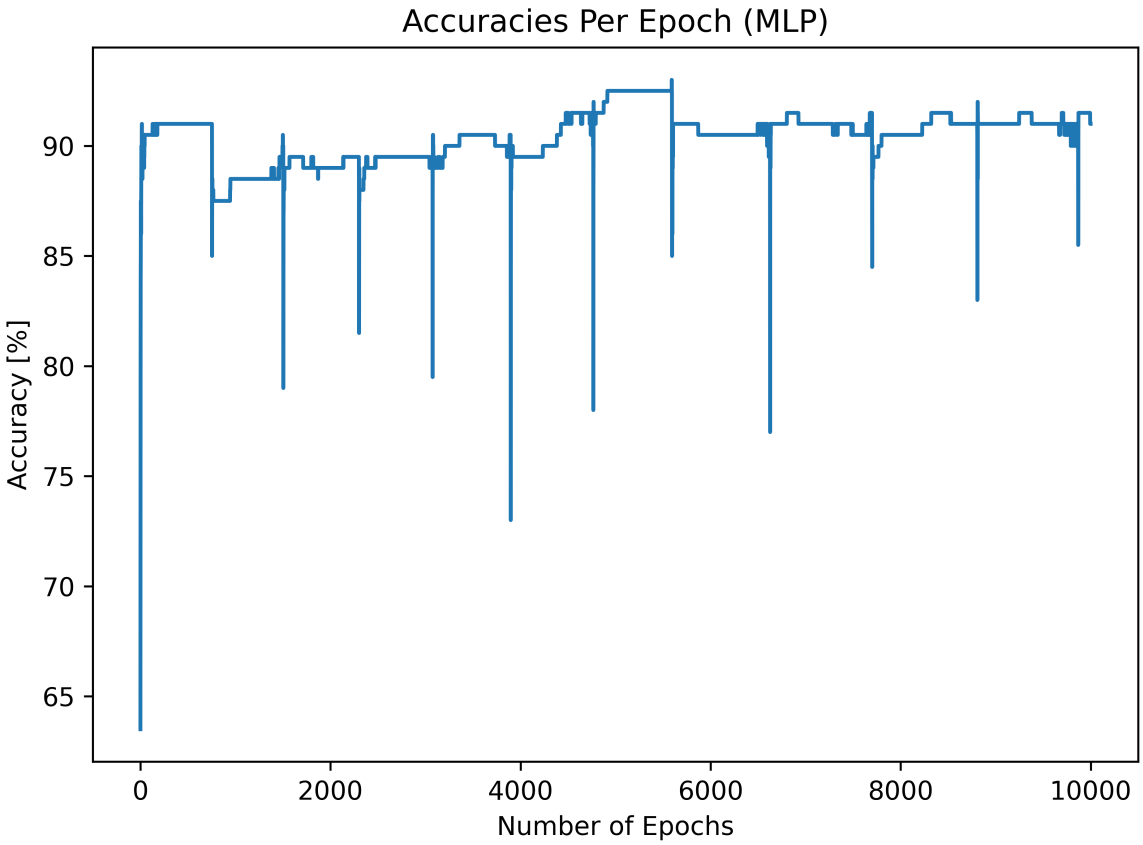
After training the model we save the model, as well as the losses per epoch and the classification accuracies per epoch. This is because training the model again each time would take a lot of time.

When plotting the losses per epoch when training the model we observe that



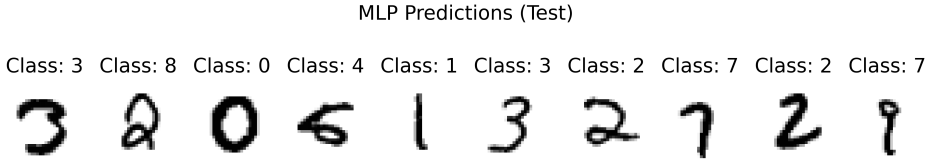
the losses clearly decrease with the number of epochs. However, the graph seems very perculiar, since after about 800 or so epochs the losses spike and then decrease again. This is very unexpected and I am not quite sure why this happens. The only guess is that this is a feature of the ADAM optimizer. I presume that when the gradient (loss) seems to decrease and decrease and we are not converging. The optimizer tries to take a large step to help get out of a minima/barren plateau. In doing so the loss increases drastically in one epoch and then starts decreasing again.

When looking at the classification accuracy per epoch we a similar trend as in the plot of the loss per epoch.



The accuracy first increases rapidly and then quite quickly seems to converge to an accuracy of about 91%. Then we see drastic drops in the accuracy at 800 epoch steps, which is likely effect due to the spikes in the losses, i.e the ADAM optimizer. For each period in between the spikes the accuracy appears to remain stable. We achieve the highest accuracy at about 93% at around 5000 epochs.

When looking at some predictions of the model we see that the model is able to predict some of the labels correctly in the test dataset but by far not all of them.



CNN

Now instead of using an MLP for image classification we will use a CNN, which is more suited for the task of image classification. A CNN has the advantage in image classification since it can learn spatial hierarchies of features. This is done by using convolutional layers, which are able to detect patterns in the image, and pooling layers, which are able to reduce the dimensionality of the image. We will use the `torch.nn.Conv2d` and `torch.nn.MaxPool2d` layers to create the CNN.

Define CNN

Our CNN will consist of two blocks that have a convolutional layer, a relu activation fucntion, and a pooling layer. The first block is defined as

```
self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, kernel_size,
                stride=stride, padding=padding
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size_max_pool)
        )
```

in the initializer of the CNN class. The second block is defined as

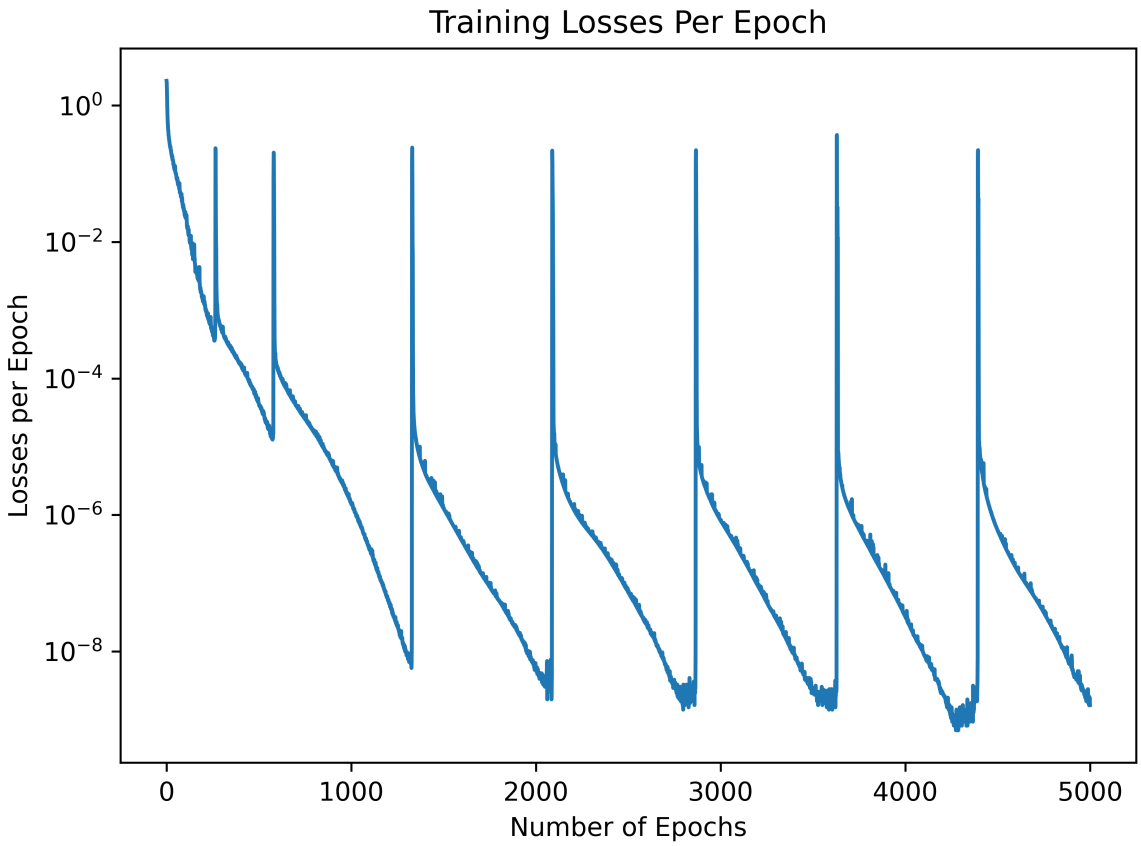
```
self.conv2 = nn.Sequential(
    nn.Conv2d(out_channels, 1, kernel_size, stride=stride, padding=padding),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size_max_pool)
)
```

Finally we add a linear layer with 10 ouputs to the CNN representing the 10 differenct classes. Now we could techincally calculate the number of inputs for the linear layer by hand, however we can let the computer also do the calculation by trying to run the model on some data and throw an error that the linear layer has the wrong number of inputs. The error message will then print the number of inputs actually needed. The linear layer is thus defined as

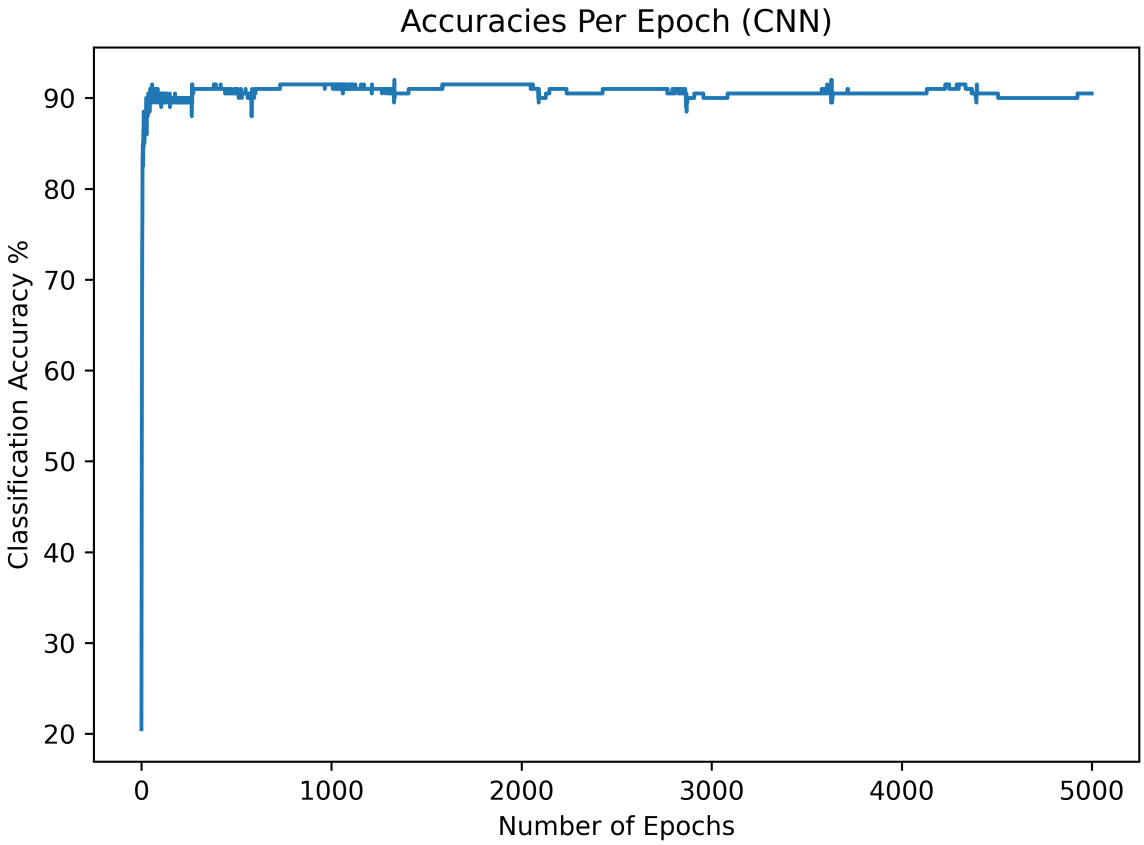
```
self.linear = Linear(49, 10)
```

Fitting

We initialise the CNN with the parameters given on the tutorial sheet. And train the model for 5000 epochs using the same training loop that was employed for the MLP. When plotting the losses per epoch we see that

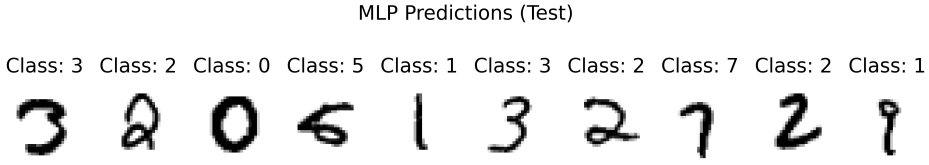


it has a similar trend to the losses of the MLP. We see similar spikes every 800 epochs or so. The losses decrease with the number of epochs, but the graph is not as smooth as one would expect. The classification accuracy per epoch is plotted as



The accuracy increases rapidly and then seems to converge to an accuracy of about 91%. The CNN converges to a high accuracy of 91% at around 100 epochs and appears to stay constantly at this accuracy for the remainder of its training. This is different to the MLP since the MLP accuracy always dropped when the losses spiked. This is not the case for the CNN.

We can look at some predictions of the CNN model of some data from the data set. We can see that the CNN model almost perfectly predicts the labels for each image



Comparison

When comparing the MLP and the CNN we have to take into account the difference of the trainable parameters for each model. The CNN has a total of 1317 trainable parameters, while the MLP has a total of 33450 trainable parameters. This is quite a significant difference. The MLP has almost 30 times as many parameters it needs to learn. Although, the MLP has many more parameters it only achieves a maximum accuracy of 93% compared to a maximum accuracy of 92% for the CNN. The CNN is able to achieve a similar accuracy with significantly fewer parameters. The CNN is thus more suited for the task of image classification than the MLP.