Deep Learning 2020 **(Due: [25-11-2020])**

Submission Assignment #3

Instructor: Jakub Tomczak *Name:* Jens van Holland, *Netid:* jhd660

When needed, the questions from part 1 and 2 are referred to as Question 1.1, Question 2.1, etc.

1 Problem statement

For this assignment, one has to build a Convolutional neural network (a CNN) and train it on the MNIST data set. The data set contains handwritten digits (graphs), these digits are values ranging from 0 to 9. Thus, there are 10 different options for a model to choose from. The training set contains 60,000 samples, the test set contains 10,000 samples. Each graph (written digit) is represented as a vector of length 784. The values of such vector are ranging from 0 to 255, indicating the intensity of a pixel.

The objective is to maximize the accuracy (and minimize the loss) which is the amount of correct predicted samples divided by the total number of samples.

2 Methodology

Firstly, the number of parameters of the CNN are calculated and compared to the number of parameters of the MLP from assignment 1. Next, the full training data set is split into two sections: the training and validation set. The training and validation set contain respectively 55,000 and 5,000 samples. Next, all the data sets are standardized using the mean (0.1307) and standard deviation (0.3081), these statistics are based on the training data set. The experiment setup:

1. Experimentation with the Adam optimizer. Based on these experiments, learning rate is set.
2. After choosing the learning rate, the influence of the initial weights on the learning process is showed. This is done with a graph.
3. At last, the final run is done with the model based on the chosen hyperparameters. The training process will be showed and the results on the test data are also given.

3 Experiments

In this section the main analysis will be explained.
In the first assignment, one made a MLP with Numpy and tried it on the MNIST dataset. The final results of the MLP are shown in table 1. These are not bad results, ofcourse there are better results on the internet.

Dataset	Final loss	Final accuracy
Training	0.465	0.909
Test	0.508	0.902

Table 1: The results of the last epoch (5) of the final run on the MNIST dataset.

(Question 1.2/1.3)
The architecture of the MLP in assignment 1 is *784 (input) - Linear(784, 300) - Sigmoid - Linear(300, 10) - Softmax*. In table 2 are the total number of parameters shown. The architecture of the CNN, analysed in this report, is *Conv(16,3) - ReLU - MaxPool(2,2) - Conv(4,3) - ReLU - MaxPool(2,2) - FC(196, 10) - Softmax*. The calculation of the total number of parameters in this CNN is shown in table 3.

Layer	Number of parameters
Linear(784,300)	$784 \cdot 300$ *(weights)* $+ 1 \cdot 300$ *(bias)* $= 235{,}500$
Linear(300,10)	$300 \cdot 10$ *(weights)* $+ 1 \cdot 10$ *(bias)* $= 3{,}010$
Total	238,510

Table 2: The calucation of the number of parameters in the MLP.

Layer	Number of parameters
Conv(16,3)	$(3 \cdot 3) \cdot 16$ *(filter parameters)* $+ 1 \cdot 16$ *(bias)* $= 160$
Conv(4,3)	$(3 \cdot 3) \cdot 16 \cdot 4$ *(filter parameters)* $+ 1 \cdot 4$ *(bias)* $= 580$
FC(196, 10)	$196 \cdot 10$ *(weights)* $+ 1 \cdot 10$ *(bias)* $= 1970$
Total	2710

Table 3: The calucation of the number of parameters in the CNN.

Due to the fact that CNN's use parameter sharing, the number of parameters in CNN's are drastically lower than in MLP's (in this case). The implementation of the CNN in Python with Pytorch is shown in appendix A. The training loop can be viewed in the script attached to this report.

(Question 2.3)
Next, one has know what learning rate is effective for the CNN model with the Adam optimizer. In figure 1 are the validation and training accuracies shown. It is interesting to see that relative low learning rates (0.00001 & 0.000001) are performing quite bad, the model is as good as guessing without seeing the image. The relative high learning rates (0.01 & 0.001) are performing quite good. The model with a learning rate of 0.0001 is not really bad, but does not perform as good as the higher learning rates. Also, it seems that there is no severe overfitting, for each model.
To summarize, the high learning rates are having better final results compared to the lower learning rates.
A batchsize of 2048 is used during each training, this is the batchsize that the available computation power could handle. Also, a learning rate decay factor $\gamma = 0.95$ is used for lowering the learning rate after each epoch.

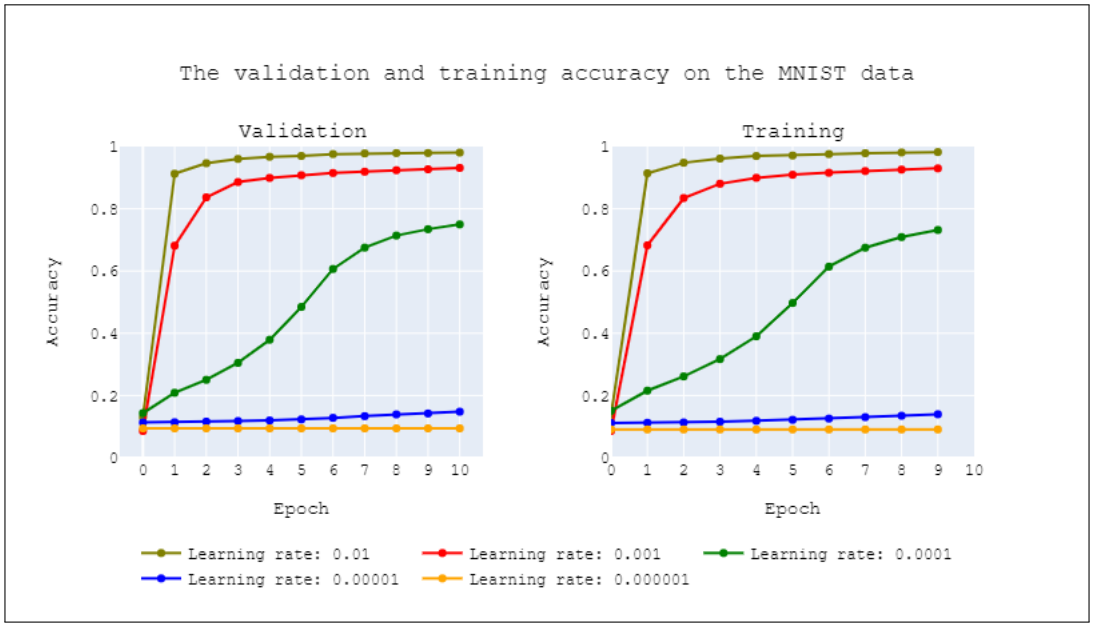

Figure 1: The validation (left) and training (right) accuracies for several learning rates on the MNIST dataset. Note that the training duration is 10 epochs, each evaluation is done before the new epoch starts.

(Question 2.2)
From figure 1, the model with learning rate 0.01 is chosen to be further examined. Next, it is good to know how constant the model's performance is. From figure 2, one can see that the training process for the three runs are pretty stable, this could be because of the high batchsize.

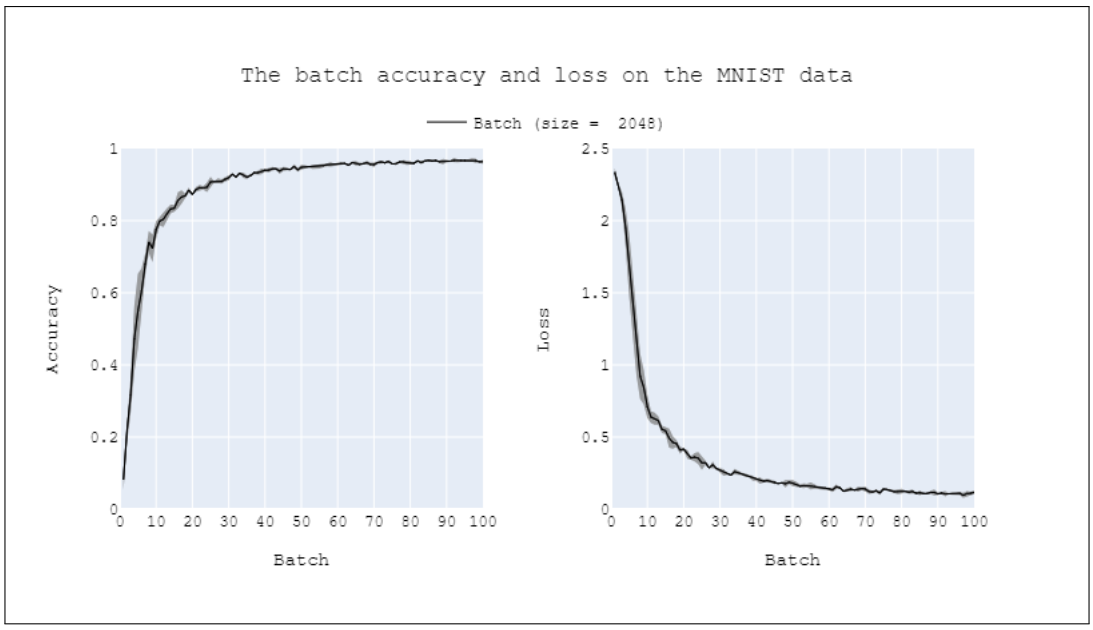

Figure 2: The spread of the batch accuracies (left) and loss (right) over three different runs.

(Question 2.1)
The final run is done with the CNN stated earlier, with the Adam optimizer, a batchsize of 2048, a learning rate of 0.01 and a learning rate decay factor of 0.95. The model is trained for 5 epochs, 10 epochs seemed to much according to figure 1. As one can see in figure 3, the training process of the model reaches its limit very fast, the performance does not increases that much after the second epoch.

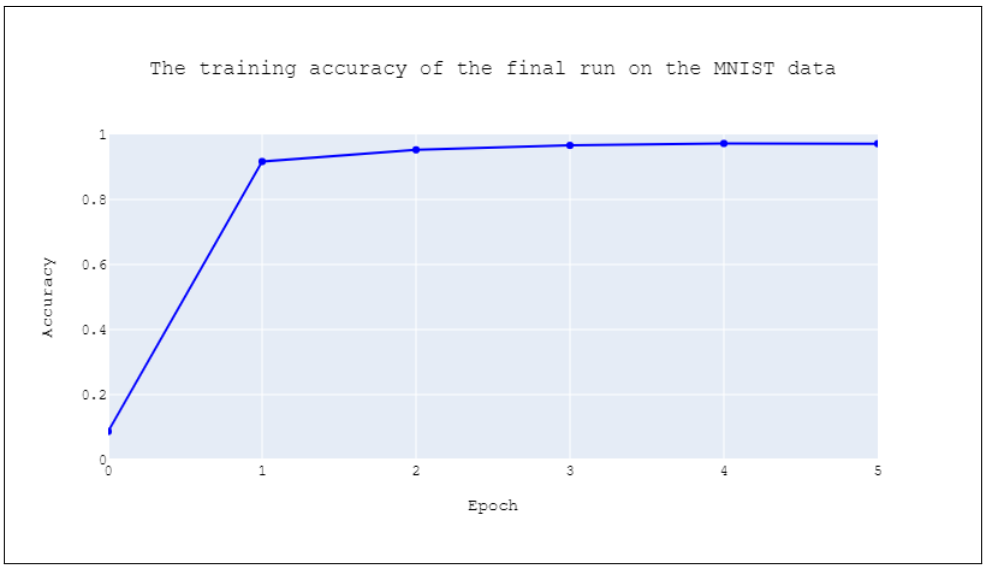

Figure 3: The accuracies of the final run on the MNIST data.

(Question 2.1)
The final accuracies are shown in figure 4, it is clear that the model performs quite well on the full training set. There is no indication of overfitting. The classification error is 1 - 0.975 = 2.5%. On average, the model will classify 2.5% of the images wrong.

Dataset	Final accuracy
Training	0.972
Test	0.975

Table 4: The results of the last epoch (5) of the final run on the MNIST dataset.

4 Results and discussion

From the experiments described above, it is clear that the CNN works really well on this dataset. Compared to the results of the MLP, table 1, it seems that the CNN outperforms the MLP. It is interesting to see that the CNN outperforms the MLP which has many more trainable parameters.
Further more, the Adam optimizer in combination with the CNN performs best with a relative high learning rate. From the experimentation, it seems like the batchsize somewhat influences the learning rate. For larger batchsizes the higher learning rates were effective and for smaller batchsizes the lower learning rates were effective.

A Implementation CNN

```
class Net(nn.Module):
    def __init__(self, model = "1"):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding= 1)
        self.pool = nn.MaxPool2d(2) # is used twice
        self.conv2 = nn.Conv2d(16, 4, 3, padding= 1)
        self.fc1 = nn.Linear(196, 10)


    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 196)
        x = self.fc1(x)
        return x

net = Net()
optimizer = optim.Adam(net.parameters(), lr = 0.01)
scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.95, last_epoch=-1, verbose=
                                             False)
net, acc_train, acc_val, running_loss, accuracy_batch = train(net, criterion, scheduler,
                                             optimizer, trainloader, valiloader)
```