Lab Course Machine Learning
Exercise Sheet 8

Prof. Dr. Dr. Lars Schmidt-Thieme, Shereen Elsayed
Information Systems and Machine Learning Lab
University of Hildesheim

January 2nd, 2022
Submission on January 14th, 2022 at 12 noon, (on learnweb, course code 3116)

Instructions

Please following these instructions for solving and submitting the exercise sheet.

1. You should submit a jupyter notebook detailing your solution.
2. Please set the seed(s) to 3116.
3. Please explain your approach i.e. how you solved a given problem and present your results in form of graphs and tables.
4. Please submit your jupyter notebook to learnweb before the deadline. Please refrain from emailing the solutions except in case of emergencies.
5. Unless explicitly noted, you are not allowed to use scikit, sklearn or any other library for solving any part.
6. Please refrain from plagiarism.

Exercise 1: Optical Character Recognition via Neural Networks (8 pts.)

The task for this exercise is to develop a Neural Network model that can classify human-written digits into either of the first 10. We shall be using concepts from the previous exercise sheets such as hyperparameter optimization and k-cross fold validation as well. Please use the Sklearn library for implementing your solution. The tasks involve:

1. Loading the MNIST digits dataset via sklearn provided built-in utility function(s).
2. Importing the necessary classes to do k-cross fold validation. You are free to choose k, depending upon your computational budget and task complexity but for most purposes 'k=5' suffices fine. Please set aside 20% of the images for *testing*.
3. Defining a hyperparameter grid for the 'MLPClassifier' that is the Neural Network model implementation from Sklearn. You need to read the documentation page to find out the hyperparameters supported.
4. Defining a *Random Search* procedure over the ranges you chose above, and then train the model by calling the *'.fit'* method for the search object.
5. Reporting one test accuracy and the best hyperparameters found.

Exercise 2: End-to-End Self-Driving via Convolutional Neural Networks (12 Points)

For this exercise there are a few background steps that need to be done before jumping to the main tasks.

1 Sign up on Kaggle.com and visit the page `https://www.kaggle.com/asrsaiteja/car-steering-angle-prediction` to see the dataset for this exercise.

2 Introduce yourself to the PyTorch library, python library for auto-differentiation and neural network modeling. The examples page can be accessed here: `https://pytorch.org/tutorials/beginner/pytorch_with_examples.html`, and the Linear layers, Convolutional layers, SGD Optimizer, Backward Gradient Calculation for Loss functions are relevant for the purpose of this exercise and should be understood.

3 Resources for understanding ConvNets:

   a. `https://www.ismll.uni-hildesheim.de/lehre/dl-20s/script/dl-05-cnn.pdf`
   b. `https://cs231n.github.io/convolutional-networks/`

2 Create a new notebook on the platform Kaggle.com (there should be button, to create such on the link from 1.), and once the data is loaded/downloaded to this notebook, follow the code snippet provided here to read/show images.

3 Divide these resulting arrays into corresponding train/validation/test splits. Leave the last 10k images for testing (images are id'ed). You are free to define the length of the validation split.

4 Implement the Convolutional Neural Network Architecture proposed in the paper titled, *"End to End Learning for Self-Driving Cars"*. The paper can be accessed here: `https://arxiv.org/abs/1604.07316`

5 Report one test RMSE for the test set of images.

Listing 1: A minimal example of a ConvNet with PyTorch

```
from torch.utils.data import DataLoader, Dataset
class Dataset(Dataset):#Inherits from torch.utils.data.Dataset
        def __init__(self):
                #default directory where data is loaded
                self.filepath = '/kaggle/input/car-steering-angle-prediction/driving_dataset/'
                self.filenames=os.listdir(self.filepath)

        def __len__(self):
                return len(self.filenames)

        def __getitem__(self, index):
                filename = self.filenames[index]
                img = cv2.imread(self.filepath+filename)
                #Resizing images to (32,32)
                resized = cv2.resize(img, (32,32), interpolation = cv2.INTER_AREA)
                #return the image converted to a numpy array its corresponding steering angle
                return torch.from_numpy(resized.transpose()).float(), torch.rand(1)

class ConvNet(torch.nn.Module):
        def __init__(self):
                super().__init__()
                self.conv1 = nn.Conv2d(3, 8, 3)
                self.lin1 = nn.Linear(7200, 1)
        def forward(self, x):
                x = self.conv1(x)
                x = x.view(x.shape[0],-1)
                x = self.lin1(x)
```

```
                return x

train_data = Dataset()
train_loader = torch.utils.data.DataLoader(train_data, batch_size=60, shuffle=True)

net = ConvNet()
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
criterion = torch.nn.MSELoss()

for i, sample_batched in enumerate(train_loader):
        optimizer.zero_grad()
        #print(i, sample_batched[0].shape, sample_batched[1].shape)
        yhat = convModel(sample_batched[0])
        loss = criterion(yhat.squeeze(), torch.rand(1))
        print("loss: ", loss.item())
        loss.backward()
        optimizer.step()
```

Bonus Exercise: Hyperparameter Tuning, Regularization with Image Transformations (3+4+3 pts.)

The aim of this exercise is to further develop modifications on top, that can hopefully lead to performance gains over the architecture from the previous exercise.

1. Tune the associated hyperparameters such as batch_size, number_of_layers, kernel_sizes, learning_rate, l1_regularization, l2_regularization coefficients etc. Either implement Random Search or Hyperband.

2. Implement the regularization scheme named "Cutout" as proposed in the paper titled, "Improved Regularization of Convolutional Neural Networks with Cutout" (landing page here: `https://arxiv.org/abs/1708.04552`).

3. Implement the regularization scheme titled, "MixUp", as proposed in the paper titled, "mixup: Beyond Empirical Risk Minimization" (landing page here: `https://arxiv.org/abs/1710.09412`).