Lab 8

Task 1. Run the code below.

Below we have an example code that performs the task of predicting a sine function using a simple fully connected three-layer network. The model takes only one x value and returns the function's predicted y value.

In the case of pytorch, the models accept a whole set (batch) of single values, which improves the model learning process.

model code: **lab2_model.py**

```
#%%
import torch

class TinyModel(torch.nn.Module):

    def __init__(self, x = 1024):
        super(TinyModel, self).__init__()

        self.activation = torch.nn.LeakyReLU()
        self.linear1 = torch.nn.Linear(1, x)
        self.linear2 = torch.nn.Linear(x, x)
        self.linear3 = torch.nn.Linear(x, 1)

    def forward(self, x):
        x = self.linear1(x)
        x = self.activation(x)
        x = self.linear2(x)
        x = self.activation(x)
        x = self.linear3(x)
        return x

if __name__=='__main__':
    tinymodel = TinyModel()

    print("model params: %i"%sum([param.nelement() for param in
model.parameters()]))

    print('The model:')
    print(tinymodel)

    print("Model params: %i"%sum([param.nelement() for param in
tinymodel.parameters()]))

    #test
    x = torch.rand(10, 1)
    print("x = ", x.size())
```

```
    y = tinymodel(x)
    print("y = ", x.size())
# %%
```

- **def __init__(self):** - constructor of our model, we define the model's layers, activation functions, etc. the order is not important.
- **def forward(self, x): -** function launched during prediction, here we control the flow of data
- **if __name__=='__main__': -** in this block we can test our model outside of the training code. We can check the correctness of the model and data flow. This code will not run when we import the file.
- **torch.nn.Linear(1, 1024)** - fully connected network layer, which accepts a data vector of size (batch,1) at the input and (batch,1024) at the output.
- **batch -** it is the number of samples that the network will analyze in one prediction, the value should not depend on the construction of the network.
- **torch.nn.LeakyReLU() -** activation function

main code**: lab2.py**

```
# init block
#%%
import torch
from lab2_model import TinyModel as Model
model = Model()

#data generator
def data_gen(batch=30, range = 2*torch.pi):
    x = torch.rand(batch,1)*range
    return x, torch.sin(x)

#test
x, y = data_gen(5)
print("x=",x)
print("y=",y)

#train block
#%%
model.train()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loss_fn = torch.nn.MSELoss(reduction="mean")
for epoche in range(30):
    err = 0
    for step in range(50):
        # Every data instance is an input + label pair
        inputs, labels = data_gen(100)
        # Zero your gradients for every batch!
        optimizer.zero_grad()
        # Make predictions for this batch
        outputs = model(inputs)
```

```
        # Compute the loss and its gradients
        loss = loss_fn(outputs, labels)
        loss.backward()
        # Model parameters optimization
        optimizer.step()

        err += loss.item()
        print("\rerror = %f "%(err), end="")
    print("\repoch= %d error= %f: \n"%(epoche,err))


#prediction/test block
#%%
model.eval()
x, y_true = data_gen(10)
y_pred = model(x)
print("     x=",x)
print("y_pred=",y_pred)
print("y_true=",y_true)
print("L1_Err=",y_true-y_pred)
```

- **#init block** - the code imports files, defines functions that generate data
- **def data_gen(batch=30, range = 2*torch.pi):** - function generate dataset for model training and testing.
- **#train block -** code that trains the model
- **torch.nn.MSELoss** - the loss function determines the error between the predicted and known data. On the basis of the error, corrections of model weights are determined
- **torch.optim.Adam -** the optimization function deals with correcting model parameters during training

Task 2.

Write a function that allows you to visualize the data (x, y) generated by the data_gen functions. Then extend it so that you can compare the outputs prediction of the model and the labels values from the data_gen function. Then display what the graph looks like after each 5th epoch during training.

Task 3.

Write a function that plots the change of errors during training. Investigate how the change of the batch parameter (it is set to 100 in the code) and epoch has an impact on the results and training process

Task 4.

Modify the model and verify how the modification affects the results. Add another layer to the model and reduce the number of parameters the layers take. Verify the results. Explore other variants. Remember that the first layer must take one parameter (x) and the last one must return one parameter (y).

Task 5.

Test your model for other data generating functions. Create a polynomial function and a nonlinear function, e.g.

$$y = \begin{cases} sin(x)\ dla\ x\ < 1 \\ 0.5x + 1\ dla\ x\ >= 1 \end{cases}$$

Task 6.

Prepare a report, one for the whole group.