Lab 9

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 from lab 8. Recently, the model was trained with randomly generated values during training. In this task, we will generate a set before training. Noise and errors that normally occur in training sets have been added to the set. **The task is to present the problem of network overfitting.** The problem occurs when the network learns individual cases “by heart”. In such cases, erroneous data and outliers have a significant impact on the trained model. We want rather our network to generalize knowledge.

main code**: lab9.py**

```
# init block
#%%
import torch
import matplotlib.pyplot as plt
from lab8_model import TinyModel as Model


model = Model(1024)


def data_plot(title, x, y, y2=None):
  fig, ax = plt.subplots()
  ax.plot(x, y, 'o')
  if y2 is not None:
      ax.plot( x, y2, 'o')
  ax.legend(title)
  plt.show()


def data_fun(batch=30, range = 2*torch.pi):
  x = torch.rand(batch,1)*range
  return x, torch.sin(x)


#test
x, y = data_fun(150)
print("x=",x)
print("y=",y)
data_plot(["Fun test"], x, y)


# dataset init block
#%%


# prepare training dataset, add noise and errors
```

```python
def dataset_gen(batch=500, range = 2*torch.pi, noise=True, errors=True):
  x, y_true = data_fun(batch)
  # add noise
  y_real = y_true.clone()
  if noise:
      y_real += torch.rand(batch, 1)*0.3-0.15
  # add errors
  y_real2 = y_real.clone()
  if errors:
      num = .15
      idx = (torch.rand(int(batch*num))*batch).long()
      y_real2[idx, :] += .1+torch.rand(int(batch*num), 1)*0.6
      idx = (torch.rand(int(batch*num))*batch).long()
      y_real2[idx, :] -= .1+torch.rand(int(batch*num), 1)*0.6
  return x, y_real2, y_true
 # dataset
# dataset = dataset_gen(noise=False,errors=False)
dataset = dataset_gen(200)
data_plot(["Real data", "Fun data"], *dataset)


#data random batch
def data_gen(dataset, batch=30):
  size = dataset[0].size(0)
  idx = (torch.rand(batch)*size).long()
  return dataset[0][idx], dataset[1][idx], dataset[2][idx]


# batch sample
data_plot(["Batch sample", "Fun"], *data_gen(dataset))
 #train block
#%%
#cuda
device = "cuda" if torch.cuda.is_available() else "cpu"
print("device: %s"%device)
model = model.to(device)

print("Train")
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
  true_error = 0
  for step in range(50):
      # Every data instance is an input + label pair
```

```
      inputs, labels, ground_true = data_gen(dataset, 100)

      inputs = inputs.to(device)
      labels = labels.to(device)
      ground_true = ground_true.to(device)
      # 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()
      true_error += loss_fn(outputs, ground_true).item()
      print("\rerror = %f , real= %f"%(err, true_error), end="")
  print("\repoch= %d error= %f, real=%f"%(epoche,err, true_error))




#eval on dataset data
#%%
model.eval()
#pred on dataset
x, y_dataset, y_true = dataset
y_pred = model(x)
eval_error = loss_fn(y_pred, y_true).item()
print("eval_dataset_error: %f"%eval_error)
data_plot(["Pred(dataset_x)", "Dataset(dataset_x)"], x, y_pred.detach(),
y_dataset)
data_plot(["Pred(dataset_x)", "Fun(dataset_x)"], x, y_pred.detach(), y_true)


#eval on random data
#%%
model.eval()
x, y_true = data_fun(dataset[0].size(0))
y_pred = model(x)
eval_error = loss_fn(y_pred, y_true).item()
print("eval_random_error: %f"%eval_error)
data_plot(["Pred(rand x)", "Fun(rand x)"], x, y_pred.detach(), y_true)
# %%
```

Task 2.

Change the number of network parameters. Investigate the effect of reducing the number of parameters of the neural network model and increasing it.

Task 3.

Investigate the effect of reducing or increasing the size of the entire **dataset**.

Task 4.

Investigate the effect of reducing or increasing the **batch** size

Task 5.

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.

Task 6.

Find the best set of parameters and shape of the neural network model.

Task 7.

Carry out exercise 6 for other data generating functions (Invent some other function).

Task 8.

Prepare a report.