•

Contents

dataset\dataset_splitpy

to top

```
lines = []
import random
with open("func_app1.csv","r") as f:
    lines = f.readlines()
random.shuffle(lines[1:])
train_lines = lines[1:351]
valid_lines = lines[351:401]
test_lines = lines[401:501]

with open("train.csv","w") as f:
    f.write(lines[0])
    for line in train_lines:
        f.write(line)

with open("validation.csv","w") as f:
    f.write(lines[0])
    for line in valid_lines:
        f.write(line)

with open("test.csv","w") as f:
    f.write(lines[0])
    for line in test_lines:
        f.write(line)
```

dataset\readmetxt

to top

```
Instructions:
1. The dataset consist of 2 attributes (features ) and 1 Target variable ('y' coloumn)
2. Use 70:10:20 ratio for dividing dataset into Training, Validation and Test datasets,
respectively
```

datasetpy

to top

```python
from torch.utils.data import Dataset
import pandas as pd
import numpy as np
import torch


class function_dataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        data = np.array(pd.read_csv(self.data_dir))
        self.input_features = data[:,0:2]
        self.target = data[:,2:]
        self.len = data.shape[0]

    def __len__(self):
        return self.len

    def __getitem__(self, index):
        features_index = torch.from_numpy(self.input_features[index])
        target_index = torch.from_numpy(self.target[index])

        return (features_index, target_index)

def test():
    train_dataset = function_dataset(data_dir='dataset/func_app1.csv')
    train_data, train_label = train_dataset[5]

    # print(train_data)
    # print(train_label)
    # print(train_data.shape)
    # print(train_label.shape)

    print(train_dataset[:][0][:,0].shape)

#test()
```

evalpy

to top

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from model import function_approximation
from dataset import function_dataset

device = 'cuda' if torch.cuda.is_available() else 'cpu'

model = function_approximation().to(device=device)
model.load_state_dict(torch.load('model_weights.pth'))

train_dataset = function_dataset("dataset/train.csv")
train_loader = DataLoader(train_dataset)

test_dataset = function_dataset("dataset/test.csv")
test_loader = DataLoader(test_dataset)

valid_dataset = function_dataset("dataset/validation.csv")
valid_loader = DataLoader(valid_dataset)


criterion = nn.MSELoss()


def accuracy(loader, model):



    avg_loss = 0
    cnt=0

    with torch.no_grad():
        for data, target in loader:
            model.eval()
            data = data.to(device=device)
            target = target.to(device=device)

            out = model(data.float())

            loss = criterion(out,target.float())
            avg_loss += loss
            cnt+=1
        avg_loss = avg_loss/cnt
        print(f"Average loss is: {avg_loss:.2f}")
        print("-----------------------------------------------------------------------------------------------")


print("Train Set metrics:")
accuracy(train_loader, model)

print("Test Set metrics:")
```

```
accuracy(test_loader, model)

print("Validation Set metrics:")
accuracy(valid_loader, model)
```

lossestxt

to top

```
6469.233530050202
6190.931101790555
5883.22599173581
5596.877008840678
5337.303012900877
5096.645053030872
4876.540628889288
4673.263784016394
4485.531428102436
4309.3988879548915
4146.353220651711
3987.0031177775136
3829.516704134464
3681.123870719509
3543.9201794371556
3414.6851106210383
3292.636105643788
3177.647691678764
3070.4610155878922
2965.9298754088622
2867.3601155916936
2774.498478832328
2694.177996311005
2603.387308492378
2521.0982459011652
2443.337388478182
2370.8838215730025
2301.8711545321453
2233.7373368663734
2169.3944282997318
2109.1946789035437
2055.541573732963
1993.0167484983224
1939.5745421948652
1887.184776655118
1839.7134871945689
1785.318972255158
1737.802955423762
1691.9880374301924
1646.6264398705248
1604.67657918612
1566.084742159608
1524.8917141796614
1488.1353378386689
1454.472732475005
1416.0973937901178
1379.5000886839684
1348.252545587967
1324.3628492451364
1284.7427391913982
1253.8237684101775
1225.3391876584703
```

```
1203.5136340186652
1177.3216696710303
1144.1754839857508
1119.3987015357463
1094.4178529508288
1080.496984217206
1054.4650020251358
1039.2800346874399
1008.5285836884219
983.8163497786051
963.5718603893727
949.1739639355894
934.6580486254738
906.531217941231
906.1871261906234
879.6276484871689
852.9956915393046
841.1800919978019
851.3361614483795
815.4862426631918
799.9278602590776
824.6835416326386
781.6744935905969
744.2075442716854
730.5548405038672
720.7871221927024
859.9738975777743
710.416381145996
683.6388357776245
671.8157706602475
724.0333896776158
750.6614222393445
638.974825054208
632.4284063589039
649.1066230627798
628.7419264671827
620.1735607577032
594.3411663187208
582.6449460217982
936.8774392072994
582.7242108148364
560.0279573015257
546.6501516908451
568.5797725175678
535.9544544666265
523.5648404347055
603.1952533671371
512.3436884481064
543.158250819989
494.1990198272382
480.4194119944332
479.88865784242785
466.2169628430863
```

```
460.3482845102381
466.6578712346125
447.8757596784238
440.71759898287075
507.3802223878073
493.0428776469179
459.53285687603557
421.46058339041474
493.49286679696957
413.3387188066945
455.0380167517763
394.3975249900304
405.37982813770515
405.66303317930965
375.8381339023152
390.4244968779601
560.762610918345
393.2744989159251
361.28605227674234
416.46362678989567
850.2836444338354
722.6960697442653
378.3849656136611
441.43901904530424
571.639129097281
446.9788434070125
355.9731980279119
336.6434778124963
502.03261507384263
466.89262110671535
324.4460891909006
330.60122164947074
319.20683654695483
330.96140365741724
315.07652046248995
295.16431202245724
301.55329312171267
314.59093438132027
283.69389822118467
289.13743150582343
285.4178505116549
282.6222133276099
266.80732124255474
267.6780870162178
291.72752652252063
301.0820093975442
255.27581384248523
259.9788112715606
248.1678358188691
247.82576465584404
249.85177773775953
238.83690804060333
237.84376075852978
```

```
234.83010477759734
232.88150086698778
241.4665825629562
238.91304998723575
257.20470013668717
372.62203589990287
493.4555190832264
393.2878895050876
385.1773704841462
233.59242927069576
254.4670273237044
245.38136153316134
255.1130062484342
492.55430691985447
233.57857808862107
253.35092944511345
389.3625830391222
216.27789609335517
208.30248116201605
211.16262579406316
207.89040443449701
206.84850883944466
198.50431550655787
194.45448658280532
198.5994727563047
198.29875357995692
198.07632111843012
183.3659305442837
186.22883623649508
182.5515711999644
180.4360689515514
187.28112202723494
181.74815717757193
175.8808969745796
186.7982012160036
175.13045971957473
181.03019590565518
164.97966405628136
181.23077370730704
164.31702040442548
210.1360731847967
166.073794739594
172.76358136821835
158.82834603458153
154.05531820874063
153.21897153900238
146.94612490718507
150.756915951739
149.56644956118734
141.00414647304362
138.33387086464742
139.9994198225728
136.26295039148678
```

```
173.0235070137456
138.81065704789154
145.9764013452654
192.2916841295122
146.28494609055886
131.42123848641722
128.50715401000951
126.5318287633781
132.42516661171416
129.2536803063485
131.93677504319106
125.83050026165675
173.22567517283525
277.35424487884666
207.19232350941746
204.27563635343614
143.7440844498987
150.70483656083766
145.87417833112747
190.9957396572195
204.24574323103099
160.02896172130264
156.06914754463227
137.03504249679838
138.5808904852336
227.88194055121357
250.47171724605732
207.51158941271257
324.22645487171803
328.7925076801918
177.52895345955326
298.28264210387397
176.5104064355126
158.30701183597859
131.1946892528449
132.04517339978605
113.07478453640728
116.37436390747283
121.26708101852888
126.69716990091605
114.60080835615871
115.01611494851193
127.01008419244408
199.93615676670373
278.1224598787306
206.48936766117055
115.21305550563017
111.66511720653399
226.1854156392861
230.7694639805426
236.73939266614767
105.8233768546375
108.96305406752832
```

```
116.28685526920437
100.96239287145195
105.75656374441792
419.12741355948947
972.8963642241725
268.9402879655027
629.9498394484898
186.00421746810633
161.92733326155226
213.0474914392311
956.3911075532483
286.42251966297096
934.7728033526901
274.79161938004165
296.7960735569876
193.5518175719151
221.68327383854893
241.31413328788426
158.70646817452382
196.18833070157513
111.0487632873666
90.87142558413387
90.80075282159515
228.1970602654754
177.42272454640047
145.02819643238092
90.32260963538347
89.57296628576437
80.61448406055611
138.2645044460545
112.82180164212893
80.8694062329891
78.86425279774637
80.29053209630429
78.92318819919888
77.58188048994475
77.21569051585854
77.27992402801107
76.06788456711014
73.36941874782963
75.94661396092152
155.516455384202
550.5940646704878
310.1773038896155
323.3200608764998
129.22442484583573
272.4147391770956
542.4246218218435
215.73482544256956
315.1827916814096
400.2342632808157
83.32092700151564
88.75267817529118
```

```
136.04832803298785
280.9024038879843
87.26981948546799
249.93659028637887
112.78914793482849
102.73537013433837
86.39022502449788
78.16423742496808
75.51665367520135
76.28393114196352
79.79160206298647
744.578666741894
93.87506908966078
409.5214490592267
162.92377699403954
85.07198117311695
180.0677148122293
118.72535401581636
70.85977097671645
63.46442570415724
84.50430408349814
65.33376987505504
65.56199990063622
71.22639788426373
80.33842978044706
91.40145357746817
117.73746535959995
65.70799887720622
63.857539637570405
84.99342494334867
61.97642273085186
60.384661950524894
58.36869345476335
```

modelpy

to top

```python
import torch
import torch.nn as nn
import torch.nn.functional as F


class function_approximation(nn.Module):
    def __init__(self):
        super(function_approximation, self).__init__()

        self.linear1 = nn.Linear(in_features=2, out_features=8, bias=True)
        self.linear2 = nn.Linear(in_features=8, out_features=4)
        self.linear3 = nn.Linear(in_features=4, out_features=1)
        self.tanh    = nn.Tanh()
        self.softmax = nn.Softmax(dim=0)


    def forward(self, x):

        x = self.tanh(self.linear1(x))
        x = self.tanh(self.linear2(x))
        x = self.linear3(x)

        return x

def test():
    model = function_approximation()
    input = torch.Tensor([4.321097794848372, 4.769609253163742])
    out = model(input)

    print(input)
    print(model)
    print(out)

#test()
```

plotpy

to top

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from dataset import function_dataset
from model import function_approximation
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd

df = pd.read_csv('dataset/func_app1.csv')

dir = "plots/"

#device = 'cuda' if torch.cuda.is_available() else 'cpu'
device = 'cpu'



train_dataset = function_dataset("dataset/train.csv")
train_loader = DataLoader(train_dataset)

test_dataset = function_dataset("dataset/test.csv")
test_loader = DataLoader(test_dataset)

valid_dataset = function_dataset("dataset/validation.csv")
valid_loader = DataLoader(valid_dataset)


def gen_plots(model,epoch):
    x1 = np.arange(0,6,0.25,dtype="float32")
    x2 = np.arange(0,6,0.25,dtype="float32")
    x1,x2 = np.meshgrid(x1,x2)

    y = np.zeros(x1.shape)

    for i in range(x1.shape[0]):
        for j in range(x1.shape[1]):

            output = model(torch.tensor([x1[i][j],x2[i][j]]))
            y[i][j]= output

    f = plt.figure()
    ax = plt.axes(projection='3d')
    surf = ax.plot_surface(x1, x2, y, cmap = cm.jet, linewidth=0, antialiased=False)
    f.colorbar(surf, shrink=0.5, aspect=10)
    ax.set_title(f'Approximated Function after {epoch} Epochs')
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    plt.savefig(dir+'epoch'+f'{epoch}'+"_approximated.png")
    plt.show()
```

```python
# Plot of loss variation with epoch
losses = []
with open("losses.txt","r") as f:
    lines = f.readlines()
    for l in lines:
        losses.append(float(l.strip()))

epochs = np.arange(1,len(losses)+1,1)
losses = np.array(losses)
f = plt.figure()
plt.title("Loss v/s Epoch")
plt.plot(epochs,losses)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.savefig(dir+"loss.png")
plt.show()
plt.close(f)


# Scatter plot of desired output v/s approximated output
model = function_approximation().to(device=device)
model.load_state_dict(torch.load('model_weights.pth'))

desired =[]
approximated = []


for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device=device)
        target = target.to(device=device)

        out = model(data.float())

        approximated.append(out.item())
        desired.append(target.item())
desired = np.array(desired)
approximated = np.array(approximated)


f = plt.figure()
plt.title("Desired v/s Approximated Scatter Plot")
plt.scatter(desired, approximated, c='b', linewidths=1)
plt.plot(desired, desired, 'r')
plt.xlabel('Desired Function')
plt.ylabel('Approximated Function')
plt.savefig(dir+"scatter.png")
plt.show()
plt.close(f)


f = plt.figure()
ax = plt.axes(projection='3d')
```

```
surf = ax.plot_trisurf(df.iloc[:,0], df.iloc[:,1], df.iloc[:,2], cmap=cm.jet,
linewidth=0, antialiased=False)
f.colorbar(surf, shrink=0.5, aspect=10)
ax.set_title(f'Desired Function')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('Desired Function')
plt.savefig(dir+"desired.png")
plt.show()


for epoch in [1,2,10,50,350]:
    model = function_approximation().to(device)
    model.load_state_dict(torch.load(f"epoch{str(epoch)}.pt",map_location=device))
    gen_plots(model,epoch)
```

READMEmd

to top

trainpy

to top

```python
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import torch.optim as optim
from dataset import function_dataset
from model import function_approximation

data_dir = 'dataset/train.csv'
device = 'cuda' if torch.cuda.is_available() else 'cpu'
batch_size = 1
learning_rate = 2e-6
epochs = 350
momentum = 0.9

train_dataset = function_dataset(data_dir)
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)

losses = []

model = function_approximation().to(device=device)

criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=momentum)


for epoch in range(epochs):
    total_loss = 0
    cnt = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data.to(device=device)
        target = target.to(device=device)

        out = model(data.float())
        loss = criterion(out, target.float())
        cnt+=1
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()

        optimizer.step()
    losses.append(total_loss/cnt)
    print(f'Epochs:{epoch+1}, Loss:{total_loss/cnt}')
    if(epoch+1 == 1 or epoch+1==2 or epoch+1==10 or epoch+1==50 or epoch+1==epochs):
        torch.save(model.state_dict(), "epoch"+str(epoch+1)+".pt")

with open("losses.txt","w") as f:
    for l in losses:
        f.write(str(l)+"\n")
torch.save(model.state_dict(), 'model_weights.pth')
```