[REG] simple_mlp_pytorch

October 3, 2025

[1]:
```
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
 ↪mean_squared_error,accuracy_score,classification_report
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
import pandas
```

[2]:
```
# CHOOSE DATASET

# Regression dataset
data = datasets.load_diabetes(as_frame=True)

X = data.data.values
y = data.target.values
X.shape
```

[2]:
```
(442, 10)
```

[3]:
```
#train test spliting
test_size=0.2
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=test_size,␣
 ↪random_state=42)
```

[4]:
```
# Standardize features
scaler=StandardScaler()
Xtr= scaler.fit_transform(Xtr)
Xte= scaler.transform(Xte)
```

[5]:
```
class MLP(nn.Module):
    def __init__(self, input_size, output_size=1, dropout_prob=0.5):
```

```
        super(MLP, self).__init__()

        self.fc1 = nn.Linear(input_size, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 64)
        self.out = nn.Linear(64, output_size)

        self.dropout = nn.Dropout(p=dropout_prob)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)

        x = F.relu(self.fc2(x))
        x = self.dropout(x)

        x = F.relu(self.fc3(x))
        x = self.dropout(x)

        x = F.relu(self.fc4(x))
        x = self.dropout(x)

        x = self.out(x)
        return x
```

[6]:
```
num_epochs=100
lr=0.0003
dropout=0.2
batch_size=64
```

[7]:
```
Xtr = torch.tensor(Xtr, dtype=torch.float32)
ytr = torch.tensor(ytr, dtype=torch.float32)
Xte = torch.tensor(Xte, dtype=torch.float32)
yte = torch.tensor(yte, dtype=torch.float32)

# Wrap Xtr and ytr into a dataset
train_dataset = TensorDataset(Xtr, ytr)

# Create DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)
```

[8]:
```
# Model, Loss, Optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = MLP(input_size=Xtr.shape[1], dropout_prob=dropout).to(device)
```

```
criterion = nn.MSELoss() #for regression
optimizer = optim.Adam(model.parameters(), lr=lr)
```

[9]:
```
# Training loop
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0

    for batch_x, batch_y in train_dataloader:
        batch_x = batch_x.to(device)
        batch_y = batch_y.to(device)

        logits = model(batch_x)
        loss = criterion(logits, batch_y.view(-1, 1))

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()

    avg_loss = epoch_loss / len(train_dataloader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")
```

```
Epoch [1/100], Loss: 29963.8867
Epoch [2/100], Loss: 29899.7767
Epoch [3/100], Loss: 29866.7109
Epoch [4/100], Loss: 29854.5889
Epoch [5/100], Loss: 28869.4707
Epoch [6/100], Loss: 29600.7695
Epoch [7/100], Loss: 29236.1309
Epoch [8/100], Loss: 29280.0104
Epoch [9/100], Loss: 29287.0186
Epoch [10/100], Loss: 29358.9215
Epoch [11/100], Loss: 29781.8675
Epoch [12/100], Loss: 28431.5807
Epoch [13/100], Loss: 28741.9020
Epoch [14/100], Loss: 28719.7448
Epoch [15/100], Loss: 27394.0889
Epoch [16/100], Loss: 26717.4395
Epoch [17/100], Loss: 25787.5550
Epoch [18/100], Loss: 23656.4255
Epoch [19/100], Loss: 22760.5400
Epoch [20/100], Loss: 20806.1188
Epoch [21/100], Loss: 18125.7946
Epoch [22/100], Loss: 15652.5057
Epoch [23/100], Loss: 12824.7303
Epoch [24/100], Loss: 10519.0179
```

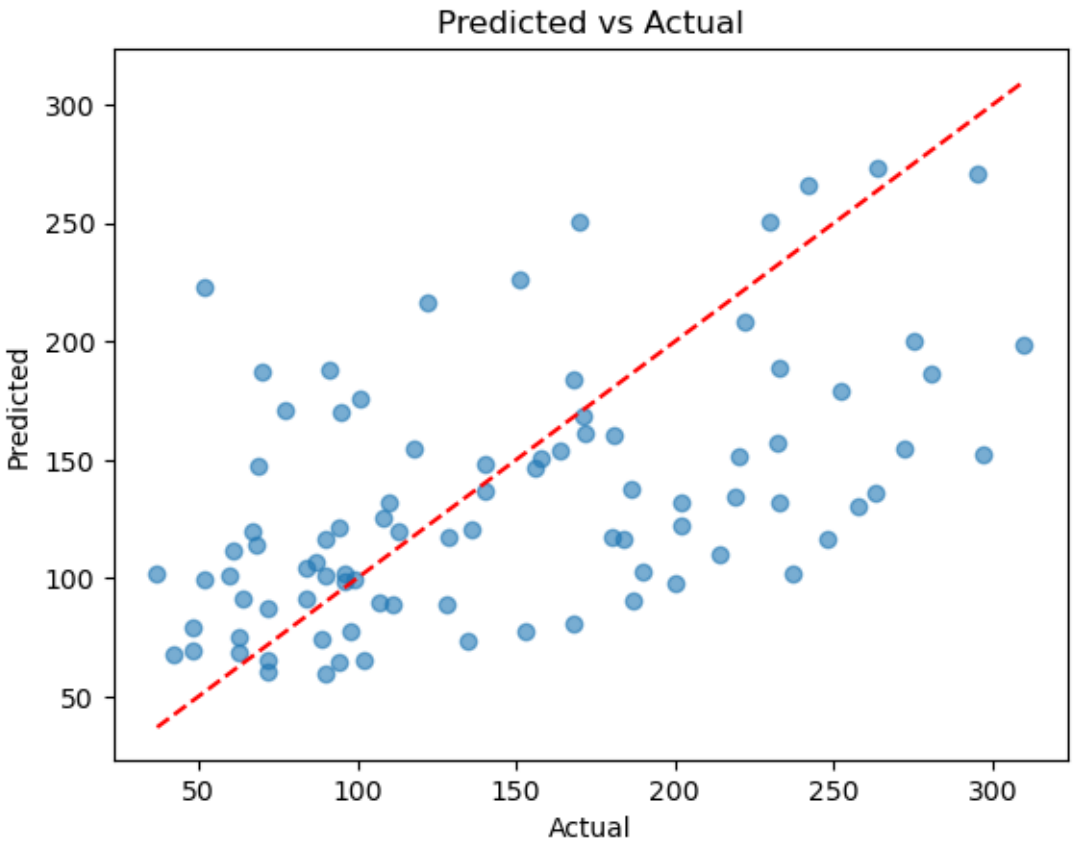
```
Epoch [25/100], Loss: 8550.2349
Epoch [26/100], Loss: 7798.2155
Epoch [27/100], Loss: 6271.0618
Epoch [28/100], Loss: 5640.2359
Epoch [29/100], Loss: 5980.7294
Epoch [30/100], Loss: 5330.4427
Epoch [31/100], Loss: 5788.4426
Epoch [32/100], Loss: 5583.0360
Epoch [33/100], Loss: 5159.8354
Epoch [34/100], Loss: 5094.7147
Epoch [35/100], Loss: 4921.0780
Epoch [36/100], Loss: 5162.0347
Epoch [37/100], Loss: 4948.4863
Epoch [38/100], Loss: 4348.9304
Epoch [39/100], Loss: 4583.8053
Epoch [40/100], Loss: 4667.6394
Epoch [41/100], Loss: 4131.9939
Epoch [42/100], Loss: 4062.7534
Epoch [43/100], Loss: 4765.8565
Epoch [44/100], Loss: 4044.8011
Epoch [45/100], Loss: 4279.3233
Epoch [46/100], Loss: 4078.8448
Epoch [47/100], Loss: 4385.0946
Epoch [48/100], Loss: 4196.5803
Epoch [49/100], Loss: 4120.9447
Epoch [50/100], Loss: 4283.8970
Epoch [51/100], Loss: 4110.0351
Epoch [52/100], Loss: 4619.4892
Epoch [53/100], Loss: 3971.0571
Epoch [54/100], Loss: 4147.9974
Epoch [55/100], Loss: 4453.0304
Epoch [56/100], Loss: 4425.9319
Epoch [57/100], Loss: 4184.6817
Epoch [58/100], Loss: 3871.1070
Epoch [59/100], Loss: 3760.2430
Epoch [60/100], Loss: 4075.9898
Epoch [61/100], Loss: 3560.8109
Epoch [62/100], Loss: 4148.9313
Epoch [63/100], Loss: 3939.6185
Epoch [64/100], Loss: 3934.4459
Epoch [65/100], Loss: 3814.6288
Epoch [66/100], Loss: 3879.1348
Epoch [67/100], Loss: 4088.5636
Epoch [68/100], Loss: 3920.8263
Epoch [69/100], Loss: 4078.7714
Epoch [70/100], Loss: 3840.1358
Epoch [71/100], Loss: 3947.9802
Epoch [72/100], Loss: 4014.3708
```

```
Epoch [73/100], Loss: 4327.6577
Epoch [74/100], Loss: 4017.3526
Epoch [75/100], Loss: 3856.8468
Epoch [76/100], Loss: 4354.9016
Epoch [77/100], Loss: 3879.2114
Epoch [78/100], Loss: 3884.2566
Epoch [79/100], Loss: 3813.7034
Epoch [80/100], Loss: 3949.5700
Epoch [81/100], Loss: 4042.1320
Epoch [82/100], Loss: 3726.5214
Epoch [83/100], Loss: 3672.1545
Epoch [84/100], Loss: 3564.3408
Epoch [85/100], Loss: 3697.4874
Epoch [86/100], Loss: 3968.1486
Epoch [87/100], Loss: 3982.2527
Epoch [88/100], Loss: 3832.3629
Epoch [89/100], Loss: 3587.4059
Epoch [90/100], Loss: 3599.8098
Epoch [91/100], Loss: 3808.5240
Epoch [92/100], Loss: 3651.7393
Epoch [93/100], Loss: 3530.6734
Epoch [94/100], Loss: 3784.5076
Epoch [95/100], Loss: 3566.1061
Epoch [96/100], Loss: 3871.3206
Epoch [97/100], Loss: 3977.1104
Epoch [98/100], Loss: 3779.8511
Epoch [99/100], Loss: 3656.1186
Epoch [100/100], Loss: 3481.5117
```

[10]:
```
y_pred=model(Xte)
print(f'MSE:{mean_squared_error(yte.detach().numpy(),y_pred.detach().
  ↪numpy())}') #regression
```

```
MSE:4207.64013671875
```

[11]:
```
plt.scatter(yte, y_pred.detach().numpy(), alpha=0.6)
plt.plot([yte.min(), yte.max()], [yte.min(), yte.max()], 'r--')
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Predicted vs Actual")
plt.show()
plt.savefig("../Plots/PredictedActualMLP.pdf", format="pdf",␣
  ↪bbox_inches="tight")
```




```
<Figure size 640x480 with 0 Axes>
```

[ ]: