Exercise

Select a regression dataset you used in a previous course and train an MLP to fit the data. Create a table comparing the results obtained in the previous course with those obtained using the MLP. A portion of the score is dedicated to achieving better performance with the MLP compared to the results from the previous course.

```
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm

import torch
from torch import nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, TensorDataset

import numpy as np
import random
np.random.seed(0)
torch.manual_seed(0)
random.seed(0)

df = pd.read_csv("Salary_dataset.csv")

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 30,\n  \"fields\": [\n{\n      \"column\": \"Unnamed: 0\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 8,\n        \"min\": 0,\n\"max\": 29,\n        \"num_unique_values\": 30,\n        \"samples\":[\n          27,\n          15,\n          23\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"YearsExperience\",\n\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":2.8378881576627184,\n        \"min\": 1.2000000000000002,\n\"max\": 10.6,\n        \"num_unique_values\": 28,\n\"samples\": [\n          4.0,\n          9.7,\n          3.8\n],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n}\n    },\n    {\n      \"column\": \"Salary\",\n      \"properties\":{\n        \"dtype\": \"number\",\n        \"std\": 27414.4297845823,\n        \"min\": 37732.0,\n        \"max\": 122392.0,\n\"num_unique_values\": 30,\n        \"samples\": [\n112636.0,\n          67939.0,\n          113813.0\n        ],\n\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
df.drop(columns=['Unnamed: 0'], inplace=True)
```

```
df.head()
```

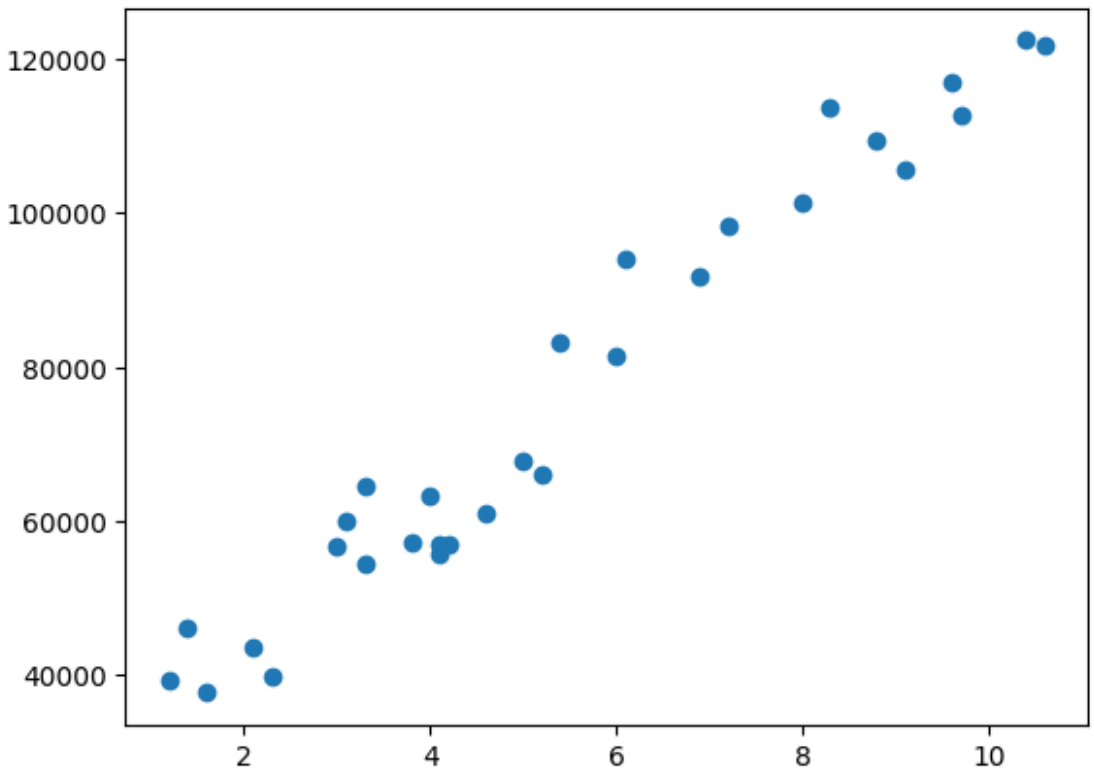

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 30,\n  \"fields\": [\n{\n      \"column\": \"YearsExperience\",\n      \"properties\": {\n\"dtype\": \"number\",\n        \"std\": 2.8378881576627184,\n\"min\": 1.2000000000000002,\n        \"max\": 10.6,\n\"num_unique_values\": 28,\n        \"samples\": [\n          4.0,\n9.7,\n          3.8\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    },\n    {\n      \"column\":\"Salary\",\n      \"properties\": {\n        \"dtype\": \"number\",\n\"std\": 27414.4297845823,\n        \"min\": 37732.0,\n\"max\": 122392.0,\n        \"num_unique_values\": 30,\n\"samples\": [\n          112636.0,\n          67939.0,\n113813.0\n        ],\n        \"semantic_type\": \"\",\n\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
df.columns
```

```
Index(['YearsExperience', 'Salary'], dtype='object')
```

```
plt.scatter(df['YearsExperience'], df['Salary'])
```

```
<matplotlib.collections.PathCollection at 0x7e53c1976f90>
```



```
class Regression_Model(nn.Module):
  def __init__(self , input_shape , output_shape):
    super().__init__()
    self.fc1 = nn.Linear(input_shape , 128)
    self.fc2 = nn.Linear(128 , 32)
    self.fc3 = nn.Linear(32 , 8)
    self.out = nn.Linear(8 , output_shape)

    self.relu = nn.ReLU()

  def forward(self , x):
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.relu(self.fc3(x))
    x = self.out(x)
    return x

from sklearn.preprocessing import StandardScaler

X = df['YearsExperience'].values.reshape(-1, 1)
Y = df['Salary'].values.reshape(-1, 1)

scaler_X = StandardScaler()
scaler_Y = StandardScaler()
X_scaled = scaler_X.fit_transform(X).astype(np.float32)
Y_scaled = scaler_Y.fit_transform(Y).astype(np.float32)

X_tensor = torch.tensor(X_scaled, dtype=torch.float32)
y_tensor = torch.tensor(Y_scaled, dtype=torch.float32).view(-1, 1)

model = Regression_Model(1 , 1)

loss_fn = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

def train_one_epoch(model, optimizer, X, Y, BS=100):
    indexes = np.random.permutation(len(X))
    losses = []

    for i, batch_start in enumerate(range(0, len(X), BS)):
        optimizer.zero_grad()

        x = X[indexes[batch_start:batch_start+BS]]
        y = Y[indexes[batch_start:batch_start+BS]]

        y_preds = model(x)

        loss = loss_fn(y_preds, y)
        loss.backward()
```

```
        optimizer.step()

        losses.append(loss.item())
    return losses

losses = []

pbar = tqdm(range(1000))
for ep in pbar:
    epoch_loss = train_one_epoch(model, optimizer, X_tensor, y_tensor)
    losses.extend(epoch_loss)

    if (ep + 1) % 100 == 0:
        pbar.set_description(f"loss={losses[-1]:0.03f}")
```

```
loss=0.015: 100%|██████████| 1000/1000 [00:01<00:00, 642.52it/s]
```

```
plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7e53c11c4190>]
```