Nama : Hurin Salimah

NIM : 1103200021

Input

```
# Show when last updated (for documentation purposes)
#mengimpor modul datetime
import datetime
#mencetak pesasn "Last updated"
print(f"Last updated: {datetime.datetime.now()}")
```

Output

```
Last updated: 2023-01-19 01:04:46.979628
```

Input

```
# Import necessary libraries
#mengimpor seluruh modul PyTorch ke dalam program python
import torch
#mengimpor modul marplotlib.pypilot dan plt mempermudah penggunaan
fungsi plotting dari matplotlib
import matplotlib.pyplot as plt
#mengimpor modul neural network dari pytorch
from torch import nn
```

Input

```
# Setup device-agnostic code
#memeriksa apakah GPU (CUDA) tersedia pada sistem
device = "cuda" if torch.cuda.is_available() else "cpu"
#mengakses nilai yang dihasilkan oleh ekpresi kondisional
device
```

Output

```
'cuda'
```

1. Create a straight line dataset using the linear regression formula (weight * X + bias).

Input

```
# Create the data parameters
#membuat data sintetis
weight = 0.3
#menentukan offset dari garis regresi
bias = 0.9
# Make X and y using linear regression feature
#mempersiapkan data input (X) yang akan menjadi fitur untuk model
regresi linear
X = torch.arange(0,1,0.01).unsqueeze(dim = 1)
#membuat data target (y) berdasarkan hubungan linear sebelumnya
y = weight * X + bias
#mencetak sampel (panjang) dari tensor X, menampilkan jumlah data yang
telah dibuat untuk fitur (X)jumlah
print(f"Number of X samples: {len(X)}")
```

```
# mencetak sampel (panjang) dari tensor y, menampilkan jumlah data yang
telah dibuat untuk fitur (y)jumlah
print(f"Number of y samples: {len(y)}")
#mencetak 10 sampel pertama dari data fitur (X) dan data target (y)
print(f"First 10 X & y samples:\nX: {X[:10]}\ny: {y[:10]}")
```

Output

```
Number of X samples: 100
Number of y samples: 100
First 10 X & y samples:
X: tensor([[0.0000],
        [0.0100],
        [0.0200],
        [0.0300],
        [0.0400],
        [0.0500],
        [0.0600],
        [0.0700],
        [0.0800],
        [0.0900]])
y: tensor([[0.9000],
        [0.9030],
        [0.9060],
        [0.9090],
        [0.9120],
        [0.9150],
        [0.9180],
        [0.9210],
        [0.9240],
        [0.9270]])
```

Input

```
# Split the data into training and testing
#menentukan titik pemisahan antara data latihan dan data uji
train_split = int(len(X) * 0.8)
#mengambil subset pertama dari data fitur (X) hingga titik
'train_split'
X_train = X[:train_split]
#mengambil subset pertama dari data fitur (y) hingga titik
'train_split'
y_train = y[:train_split]
#mengambil subset pertama dari data fitur (X) mulai dari titik
'train_split' hingga akhir data (X)
X_test = X[train_split:]
#mengambil subset pertama dari data fitur (y) mulai dari titik
'train_split' hingga akhir data (y)
y_test = y[train_split:]
#mencetak panjang dari 'X_train', y_train', 'X_test', dan 'y_test'
len(X_train),len(y_train),len(X_test),len(y_test)
```

Output

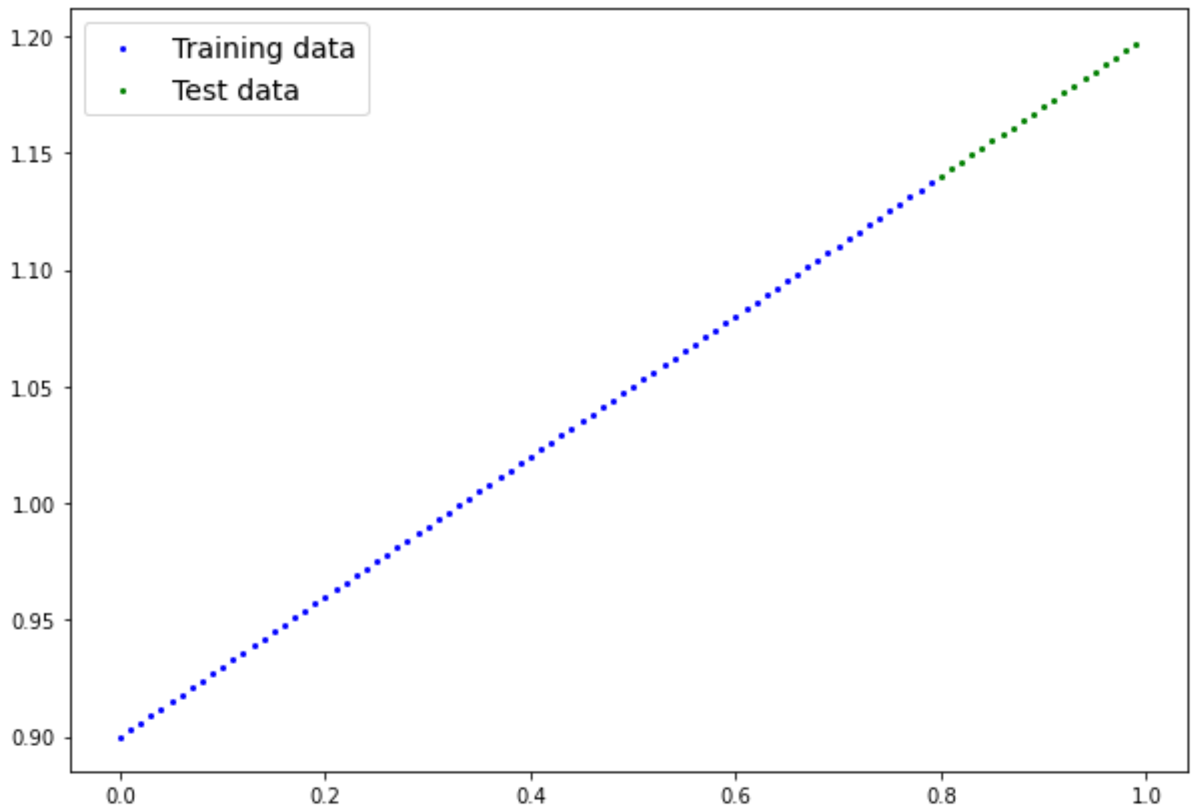
(80, 80, 20, 20)

Input

```
# Plot the training and testing data
def plot_predictions(train_data = X_train,
                     train_labels = y_train,
                     test_data = X_test,
                     test_labels = y_test,
                     predictions = None):
#membuat sebuah figure
  plt.figure(figsize = (10,7))
#membuat scatter plot dari data latihan
  plt.scatter(train_data,train_labels,c = 'b',s = 4,label = "Training
data")
#membuat scatter plot dari data uji
  plt.scatter(test_data,test_labels,c = 'g',s = 4,label = "Test data")

#memeriksa argumen
  if predictions is not None:
#menambahkan scatter plot pada data uji
    plt.scatter(test_data,predictions,c = 'r',s = 4,label =
"Predictions")
#menambahkan legenda ke plot dengan ukuran teks 14
  plt.legend(prop = {"size" : 14})
#memanggil tanpa argumen, secara default akan menggunakan data latihan
dan data uji
plot_predictions()
```

Output



2. Build a PyTorch model by subclassing nn.Module.

Input

```
# Create PyTorch linear regression model by subclassing nn.Module
## Option 1
class LinearRegressionModel(nn.Module):
#mendefinisikan 2 parameter
  def __init__(self):
    super().__init__()
    self.weight = nn.Parameter(data=torch.randn(1,
                                                requires_grad=True,
                                                dtype=torch.float
                                                ))

    self.bias = nn.Parameter(data=torch.randn(1,
                                              requires_grad=True,
                                              dtype=torch.float
                                              ))
#mendefinisikan proses perhitungan forward pass pada model
  def forward(self, x):
    return self.weight * x + self.bias

torch.manual_seed(42)
model_1 = LinearRegressionModel()
model_1,model_1.state_dict()
```

Output

(LinearRegressionModel(),
OrderedDict([('weight', tensor([0.3367])), ('bias', tensor([0.1288]))]))

Input

```
#mengambil iterator pertama dari parameter model
next(model_1.parameters()).device
```

Output

device(type='cpu')

Input

```
# Instantiate the model and put it to the target device
#memindahkan model ke perangkat yang telah ditentukan
model_1.to(device)
#memberikan daftar parameter yang dipindahkan
list(model_1.parameters())
```

Output

[Parameter containing:
tensor([0.3367], device='cuda:0', requires_grad=True), Parameter containing:
tensor([0.1288], device='cuda:0', requires_grad=True)]

3. Create a loss function and optimizer using nn.L1Loss() and torch.optim.SGD(params, lr) respectively.

Input

```
#menghitung mean absolute error (MAE) antara prediksi dan target
```

```
loss_fn = nn.L1Loss()
#paramater akan di optimalkan oleh optimazer
optimizer = torch.optim.SGD(params = model_1.parameters(),
                            lr = 0.01)
```

Input

```
# Training loop
# Train model for 300 epochs
torch.manual_seed(42)

epochs = 300

# Send data to target device
X_train = X_train.to(device)
X_test = X_test.to(device)
y_train = y_train.to(device)
y_test = y_test.to(device)

for epoch in range(epochs):
  ### Training

  # Put model in train mode
  model_1.train()

  # 1. Forward pass
  y_pred = model_1(X_train)

  # 2. Calculate loss
  loss = loss_fn(y_pred,y_train)

  # 3. Zero gradients
  optimizer.zero_grad()

  # 4. Backpropagation
  loss.backward()

  # 5. Step the optimizer
  optimizer.step()

  ### Perform testing every 20 epochs
  if epoch % 20 == 0:
    # Put model in evaluation mode and setup inference context
    model_1.eval()
    with torch.inference_mode():
      # 1. Forward pass
      y_preds = model_1(X_test)
      # 2. Calculate test loss
      test_loss = loss_fn(y_preds,y_test)
      # Print out what's happening
```

```
      print(f"Epoch: {epoch} | Train loss: {loss:.3f} | Test loss:
{test_loss:.3f}")
```

Output

```
Epoch: 0 | Train loss: 0.757 | Test loss: 0.725
Epoch: 20 | Train loss: 0.525 | Test loss: 0.454
Epoch: 40 | Train loss: 0.294 | Test loss: 0.183
Epoch: 60 | Train loss: 0.077 | Test loss: 0.073
Epoch: 80 | Train loss: 0.053 | Test loss: 0.116
Epoch: 100 | Train loss: 0.046 | Test loss: 0.105
Epoch: 120 | Train loss: 0.039 | Test loss: 0.089
Epoch: 140 | Train loss: 0.032 | Test loss: 0.074
Epoch: 160 | Train loss: 0.025 | Test loss: 0.058
Epoch: 180 | Train loss: 0.018 | Test loss: 0.042
Epoch: 200 | Train loss: 0.011 | Test loss: 0.026
Epoch: 220 | Train loss: 0.004 | Test loss: 0.009
Epoch: 240 | Train loss: 0.004 | Test loss: 0.006
Epoch: 260 | Train loss: 0.004 | Test loss: 0.006
Epoch: 280 | Train loss: 0.004 | Test loss: 0.006
```

4. Make predictions with the trained model on the test data.

Input

```
# Make predictions with the model
#mengubah model ke mode evaluasi
model_1.eval()

#melakukan prediksi pada data uji
with torch.inference_mode():
#melakukan prediksi dengan model pada data uji
  y_preds = model_1(X_test)
#hasil prediksi dari model pada dataa uji
y_preds
```

Output

```
tensor([[1.1464],
     [1.1495],
     [1.1525],
     [1.1556],
     [1.1587],
     [1.1617],
     [1.1648],
     [1.1679],
     [1.1709],
     [1.1740],
     [1.1771],
     [1.1801],
     [1.1832],
     [1.1863],
     [1.1893],
     [1.1924],
     [1.1955],
     [1.1985],
     [1.2016],
     [1.2047]], device='cuda:0')
```

Input

```
#mengubah tensor yang berada di CPU
y_preds.cpu()
```

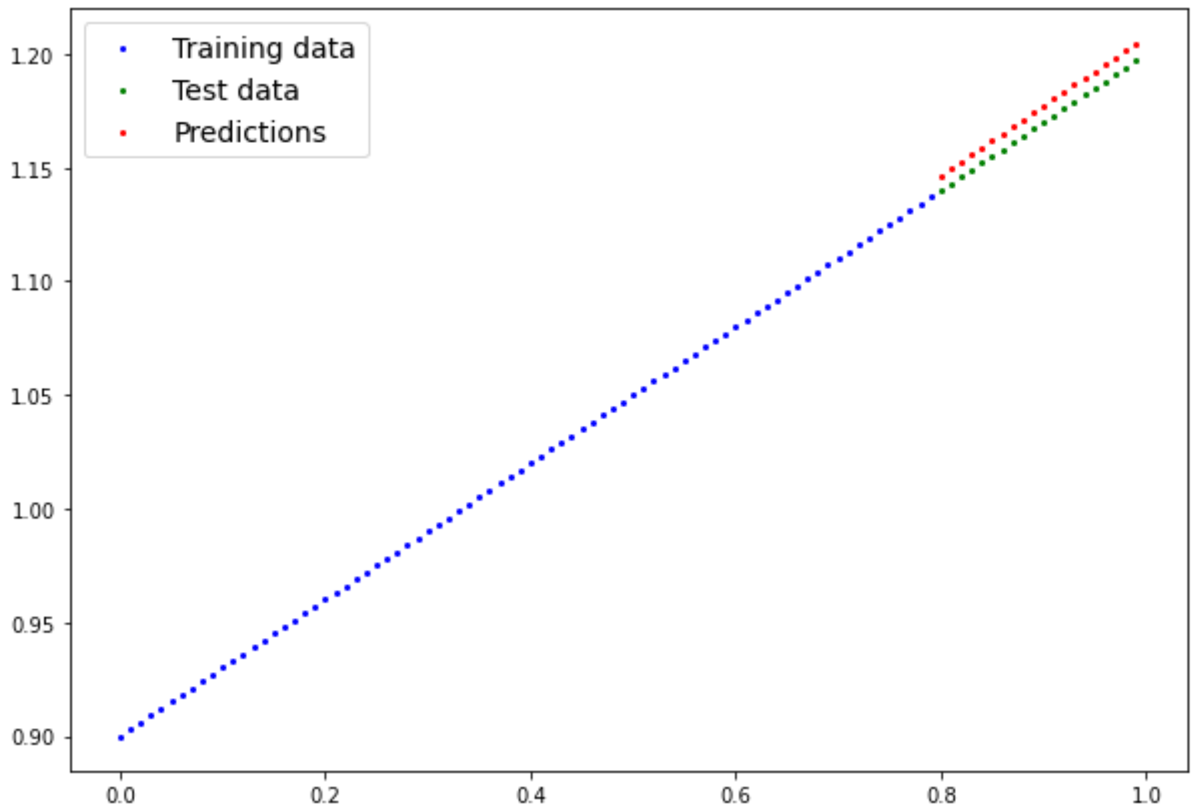
Output

```
tensor([[1.1464],
        [1.1495],
        [1.1525],
        [1.1556],
        [1.1587],
        [1.1617],
        [1.1648],
        [1.1679],
        [1.1709],
        [1.1740],
        [1.1771],
        [1.1801],
        [1.1832],
        [1.1863],
        [1.1893],
        [1.1924],
        [1.1955],
        [1.1985],
        [1.2016],
        [1.2047]])
```

Input

```
# Plot the predictions (these may need to be on a specific device)
plot_predictions(predictions = y_preds.cpu())
```

Output



5. Save your trained model's state_dict() to file.

Input

```
from pathlib import Path

# 1. Create models directory
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents = True,exist_ok = True)
# 2. Create model save path
MODEL_NAME = "01_pytorch_model"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
# 3. Save the model state dict
print(f"Saving model to {MODEL_SAVE_PATH}")
torch.save(obj = model_1.state_dict(),f = MODEL_SAVE_PATH)
```

Output

```
Saving model to models/01_pytorch_model
```

Input

```
# Create new instance of model and load saved state dict (make sure to
put it on the target device)
#memastikan kode sudah tersimpan sesuai dengan model sebelumnya
loaded_model = LinearRegressionModel()
#memastika berisi PATH yang tepat dari model
loaded_model.load_state_dict(torch.load(f = MODEL_SAVE_PATH))
#memastikan perangkat sudah sesuai
loaded_model.to(device)
```

Output

LinearRegressionModel()

Input

```
# Make predictions with loaded model and compare them to the previous
#mengevaluasi perbedaan absolut antara dua tensor
y_preds_new = loaded_model(X_test)
#mencetak hasil perbandingan
y_preds == y_preds_new
```

Output

```
tensor([[True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
        [True],
```

```
        [True],
        [True],
        [True],
        [True]], device='cuda:0')
```

Input

```
#mengembalikan state dictionary
loaded_model.state_dict()
```

Output

```
OrderedDict([('weight', tensor([0.3067], device='cuda:0')),
             ('bias', tensor([0.9011], device='cuda:0'))])
```