01. PyTorch Workflow Exercise

Nama: Dilara Kynta Putri

NIML 1103204059

Kelas: TK44G4

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

```
    Last updated: 2024-01-03 15:20:24.607539
```

kode tersebut mencetak tanggal dan waktu saat ini sebagai informasi terakhir pembaruan (last updated) pada dokumentasi

```
# Import necessary libraries
import torch
import matplotlib.pyplot as plt
from torch import nn
```

kode tersebut berfungsi untuk mengimpor beberapa pustaka yang diperlukan dalam pengembangan dan pelatihan model menggunakan PyTorch

```
# Setup device-agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
    'cuda'
```

kode tersebut digunakan untuk menentukan perangkat (device) yang akan digunakan selama pelatihan model. output yang dihasilkan yaitu 'cuda' yang artinya GPU tersedia pada sistem

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

- Set `weight=0.3` and `bias=0.9` there should be at least 100 datapoints total.
- Split the data into 80% training, 20% testing.
- Plot the training and testing data so it becomes visual.

Your output of the below cell should look something like:

```
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]])
```

Of course the numbers in `X` and `y` may be different but ideally they're created using the linear regression formula.

```
# Create the data parameters
weight = 0.3
bias = 0.9
# Make X and y using linear regression feature
X = torch.arange(0,1,0.01).unsqueeze(dim = 1)
y = weight * X + bias
print(f"Number of X samples: {len(X)}")
print(f"Number of y samples: {len(y)}")
print(f"First 10 X & y samples:\nX: {X[:10]}\ny: {y[:10]}")
```

```
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]])
```

kode tersebut digunakan untuk membuat parameter data dan menghasilkan dataset sintetis menggunakan model regresi linear. Hasilnya adalah dataset sintetis yang terdiri dari pasangan input (X) dan output (y) yang dihasilkan menggunakan model regresi linear dengan bobot 0.3 dan bias 0.9. Setiap elemen X adalah nilai dari 0 hingga 1 dengan interval 0.01, dan setiap elemen y dihasilkan dari model regresi linear yang diterapkan pada setiap elemen X.

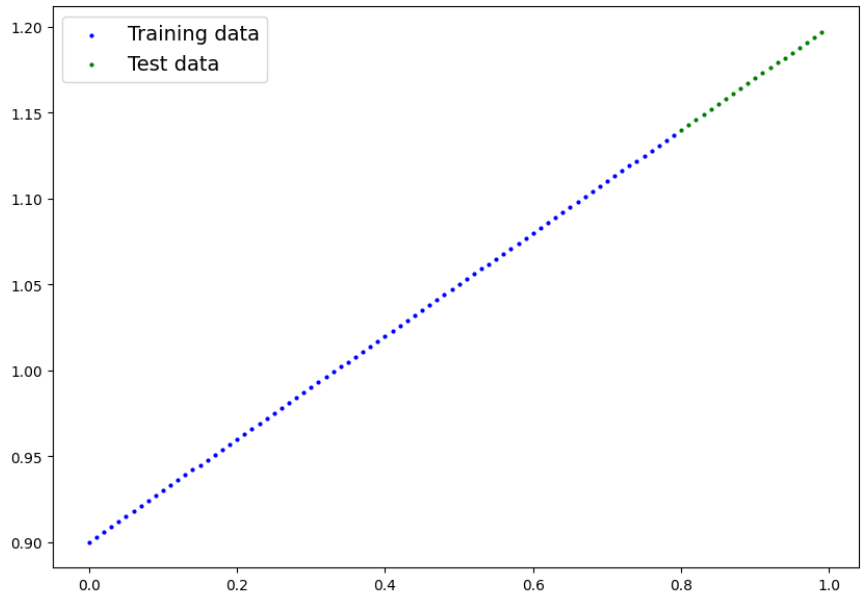
```
# Split the data into training and testing
train_split = int(len(X) * 0.8)
X_train = X[:train_split]
y_train = y[:train_split]
X_test = X[train_split:]
y_test = y[train_split:]
len(X_train),len(y_train),len(X_test),len(y_test)
```

```
(80, 80, 20, 20)
```

kode tersebut digunakan untuk membagi dataset menjadi subset pelatihan dan pengujian. hasil outputnya tersebut menunjukkan informasi panjang jumlah sampel dari setiap subset pelatihan dan pengujian

```
# 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):
  plt.figure(figsize = (10,7))
  plt.scatter(train_data,train_labels,c = 'b',s = 4,label = "Training data")
  plt.scatter(test_data,test_labels,c = 'g',s = 4,label = "Test data")

  if predictions is not None:
    plt.scatter(test_data,predictions,c = 'r',s = 4,label = "Predictions")
  plt.legend(prop = {"size" : 14})
plot_predictions()
```



kode tersebut mendefinisikan fungsi plot_predictions untuk memvisualisasikan data pelatihan dan pengujian, serta prediksi model jika disediakan. Hasil outputnya adalah scatter plot yang menampilkan data pelatihan (biru), data pengujian (hijau), dan jika diberikan, prediksi model (merah). Saat ini, karena tidak ada prediksi yang disediakan (parameter predictions adalah None), plot hanya menampilkan data pelatihan dan pengujian.

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

- Inside should be a randomly initialized `nn.Parameter()` with `requires_grad=True`, one for `weights` and one for `bias`.
- Implement the `forward()` method to compute the linear regression function you used to create the dataset in 1.
- Once you've constructed the model, make an instance of it and check its `state_dict()`.
- **Note:** If you'd like to use `nn.Linear()` instead of `nn.Parameter()` you can.

```
# Create PyTorch linear regression model by subclassing nn.Module
## Option 1
class LinearRegressionModel(nn.Module):
  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
                                          ))

  def forward(self, x):
    return self.weight * x + self.bias

# ## Option 2
# class LinearRegressionModel(nn.Module):
#   def __init__(self):
#     super().__init__()
#     self.linear_layer = nn.Linear(in_features = 1,
#                                   out_features = 1)
#   def forward(self,x : torch.Tensor) -> torch.Tensor:
#     return self.linear_layer(x)

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

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

kode tersebut mendefinisikan model regresi linear dalam PyTorch menggunakan dua opsi yang berbeda, dan kemudian menginisialisasi instance dari model.

```
next(model_1.parameters()).device

     device(type='cpu')
```

kode tersebut digunakan untuk mendapatkan perangkat (device) pada sejumlah parameter pertama dari model model_1.

```
# Instantiate the model and put it to the target device
model_1.to(device)
list(model_1.parameters())

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

kode tersebut digunakan untuk menginisialisasi model model_1 dan memindahkannya ke perangkat target yang telah ditentukan sebelumnya. Hasil output dari list(model_1.parameters()) menunjukkan daftar parameter-parameter model beserta perangkat tempat mereka disimpan (CPU atau GPU).

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

- Set the learning rate of the optimizer to be 0.01 and the parameters to optimize should be the model parameters from the model you created in 2.
- Write a training loop to perform the appropriate training steps for 300 epochs.
- The training loop should test the model on the test dataset every 20 epochs.

```
# Create the loss function and optimizer
loss_fn = nn.L1Loss()
optimizer = torch.optim.SGD(params = model_1.parameters(),
                            lr = 0.01)
```

kode tersebut digunakan untuk membuat fungsi loss dan optimizer yang akan digunakan selama pelatihan model.

```
# 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}")
```

```
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
```

kode tersebut merupakan sebuah training loop yang digunakan untuk melatih model regresi linear (model_1) selama 300 epochs (iterasi melalui seluruh dataset). Proses ini terus berlanjut hingga selesai seluruh epochs, dan setiap 20 epochs akan mencetak informasi loss pada data pelatihan dan pengujian.

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

- Visualize these predictions against the original training and testing data (**note:** you may need to make sure the predictions are *not* on the GPU if you want to use non-CUDA-enabled libraries such as matplotlib to plot).

```
# Make predictions with the model
model_1.eval()

with torch.inference_mode():
  y_preds = model_1(X_test)
y_preds
```

```
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')
```

kode tersebut digunakan untuk membuat prediksi dengan model regresi linear (model_1) pada data pengujian (X_test). output yang dihasilkan yaitu berisi nilai prediksi tensor untuk setiap sampel dalam data pengujian

```
y_preds.cpu()
```

```
    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]])
```

kode y_preds.cpu() digunakan untuk memindahkan tensor y_preds (tensor hasil prediksi sebelumnya) dari perangkat GPU ke CPU. Dengan menggunakan y_preds.cpu(), didapatkan tensor yang setara dengan y_preds, tetapi sekarang berada di CPU

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

kode plot_predictions(predictions=y_preds.cpu()) digunakan untuk memvisualisasikan hasil prediksi yang telah dipindahkan ke CPU

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

- Create a new instance of your model class you made in 2. and load in the `state_dict()` you just saved to it.
- Perform predictions on your test data with the loaded model and confirm they match the original model predictions from 4.

```
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)
```

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

kode tersebut digunakan untuk menyimpan model PyTorch ke dalam file menggunakan torch.save()

```
# Create new instance of model and load saved state dict (make sure to put it on the target device)
loaded_model = LinearRegressionModel()
loaded_model.load_state_dict(torch.load(f = MODEL_SAVE_PATH))
loaded_model.to(device)
```

```
    LinearRegressionModel()
```

ode tersebut digunakan untuk membuat instance baru dari model linear (LinearRegressionModel) dan memuat state dictionary dari model yang telah disimpan sebelumnya.

```
# Make predictions with loaded model and compare them to the previous
y_preds_new = loaded_model(X_test)
y_preds == y_preds_new
```

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