Nama : Nabilah Salwa

NIM : 1103204060

UAS Machine Learning

01. PyTorch Workflow Exercise

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

Kode di atas menggunakan library `datetime` untuk mencetak informasi terkait waktu terakhir pembaruan. Dengan menggunakan `datetime.datetime.now()`, waktu saat ini diambil dan dicetak dalam format yang sesuai. Tujuan dari kode ini adalah untuk menyediakan informasi timestamp atau cap waktu ketika kode terakhir kali diperbarui. Hal ini bermanfaat dalam konteks dokumentasi untuk melacak waktu terakhir perubahan atau pembaruan kode tersebut.

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

Kode di atas dimulai dengan mengimpor beberapa library yang diperlukan untuk penggunaan PyTorch dan matplotlib. Pertama, `import torch` digunakan untuk mengimpor library PyTorch yang merupakan framework deep learning. Selanjutnya, `import matplotlib.pyplot as plt` digunakan untuk mengimpor modul pyplot dari matplotlib, yang memungkinkan visualisasi data dengan grafik dan plot. Terakhir, `from torch import nn` digunakan untuk mengimpor modul `nn` dari PyTorch, yang menyediakan kelas-kelas untuk pembuatan dan pelatihan model neural network. Keseluruhan, kode ini menyiapkan lingkungan kerja dengan mengimpor library PyTorch dan matplotlib, serta kelas-kelas yang diperlukan untuk pembangunan model neural network.

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

Kode di atas bertujuan untuk menentukan perangkat (device) yang akan digunakan dalam eksekusi kode, dengan memeriksa ketersediaan GPU. Pada baris pertama, `device = "cuda" if torch.cuda.is_available() else "cpu"` digunakan untuk menetapkan nilai variabel `device` menjadi

"cuda" jika GPU tersedia, dan "cpu" jika tidak. Kemudian, nilai dari `device` dicetak untuk memberikan informasi mengenai perangkat yang akan digunakan dalam eksekusi selanjutnya. Dengan pendekatan ini, kode dapat dijalankan dengan baik baik pada CPU maupun GPU, memberikan fleksibilitas dalam penggunaan perangkat keras.

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

```
# 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]}")
```

Kode di atas membuat parameter-parameter data untuk suatu model regresi linear. Nilai bobot (weight) diatur menjadi 0.3 dan nilai bias diatur menjadi 0.9. Selanjutnya, menggunakan konsep regresi linear, dibuat pasangan data input-output (X dan y). Tensor X dibuat dengan menggunakan `torch.arange(0, 1, 0.01)` yang menghasilkan nilai-nilai dari 0 hingga 0.99 dengan langkah 0.01, kemudian diubah bentuknya dengan `unsqueeze(dim=1)` untuk mendapatkan dimensi kolom. Tensor y kemudian dihasilkan dengan menggunakan persamaan regresi linear sederhana: y = weight * X + bias.

Hasilnya, kode mencetak informasi tentang jumlah sampel dalam X dan y, serta 10 sampel pertama dari X dan y. Informasi ini memberikan gambaran awal tentang data yang telah dibuat, membentuk dasar untuk pelatihan model regresi linear.

```
# 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)
```

Kode di atas bertujuan untuk membagi data menjadi set pelatihan (training) dan pengujian (testing) untuk keperluan evaluasi model. Pertama, variabel `train_split` dihitung sebagai 80% dari jumlah total sampel dalam data X. Selanjutnya, data X dan y dibagi berdasarkan nilai `train_split`. Bagian pertama (80% pertama) digunakan sebagai data pelatihan (`X_train` dan

`y_train`), sedangkan bagian sisanya (20% terakhir) digunakan sebagai data pengujian (`X_test` dan `y_test`).

Kemudian, kode mencetak panjang (jumlah sampel) dari masing-masing set data pelatihan dan pengujian. Dengan demikian, proses ini memastikan bahwa kita memiliki dua set data terpisah untuk pelatihan dan pengujian, yang diperlukan dalam pengembangan dan evaluasi model machine learning. Panjang set data tersebut memberikan gambaran tentang ukuran masing-masing set, yang penting untuk pemahaman lebih lanjut terkait ukuran dan distribusi data yang digunakan.

```
# 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 di atas mendefinisikan sebuah fungsi `plot_predictions` untuk menghasilkan visualisasi data pelatihan, data pengujian, dan prediksi model. Fungsi ini menggunakan matplotlib untuk membuat scatter plot yang menunjukkan titik-titik data pelatihan (berwarna biru), data pengujian (berwarna hijau), dan prediksi model jika ada (berwarna merah).

Pada awalnya, ukuran figur (figure size) diatur menjadi 10x7 dengan `plt.figure(figsize=(10,7))`. Selanjutnya, menggunakan `plt.scatter()`, titik-titik data pelatihan dan pengujian ditampilkan pada grafik dengan warna yang berbeda. Jika ada prediksi model yang disertakan, titik-titik prediksi juga ditampilkan pada grafik dengan warna merah.

Terakhir, `plt.legend()` digunakan untuk menambahkan legenda ke grafik dengan label yang sesuai untuk setiap jenis data (pelatihan, pengujian, dan prediksi). Hasilnya adalah visualisasi yang memberikan gambaran tentang distribusi dan hubungan antara data pelatihan, data pengujian, dan prediksi model.

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

```
# Create PyTorch linear regression model by subclassing nn.Module
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

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

Kode di atas menunjukkan pembuatan model regresi linear menggunakan PyTorch dengan menerapkan subclassing dari kelas `nn.Module`. Model tersebut disebut `LinearRegressionModel`. Pada metode konstruktor (`__init__`), parameter-pesos model (`weight` dan `bias`) diinisialisasi sebagai objek `nn.Parameter` dengan nilai acak yang dihasilkan oleh `torch.randn`. Penggunaan `requires_grad=True` menandakan bahwa perubahan nilai parameter akan diperhitungkan selama proses pelatihan model, dan `dtype=torch.float` menetapkan tipe data parameter sebagai float. Metode `forward` digunakan untuk mendefinisikan langkah-langkah perhitungan saat melakukan inferensi model. Dalam hal ini, model regresi linear sederhana diimplementasikan dengan rumus matematis `self.weight * x + self.bias`, di mana `x` adalah input.

Selanjutnya, dengan menggunakan `torch.manual_seed(42)`, seed acak ditetapkan untuk memastikan reproduktibilitas hasil acak pada saat inisialisasi model. Model `model_1` dibuat sebagai instance dari `LinearRegressionModel`. Fungsi `state_dict()` digunakan untuk mengakses dan mencetak state dictionary dari model, yang berisi nilai-nilai dari semua parameter-pesos dan informasi lainnya. Keseluruhan, kode ini menciptakan dan menginisialisasi model regresi linear dengan PyTorch menggunakan subclassing dari `nn.Module`.

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

Kode di atas bertujuan untuk mengakses perangkat (device) tempat parameter pertama dari model `model_1` disimpan. Dengan menggunakan `model_1.parameters()`, kita mendapatkan iterator yang berisi semua parameter-pesos dari model. Pada pemanggilan `next(...)`, kita mengambil parameter pertama dari iterator tersebut. Kemudian, dengan menggunakan `.device`, kita mendapatkan informasi tentang perangkat (device) di mana parameter tersebut disimpan.

Hasilnya adalah perangkat (device) tempat parameter pertama dari model `model_1` disimpan. Informasi ini memberikan gambaran tentang lokasi di mana perhitungan model akan dilakukan, yang penting terutama ketika model dijalankan pada GPU atau CPU yang berbeda.

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

Kode di atas bertujuan untuk menginstansiasi model regresi linear (`model_1`) dan memindahkannya ke perangkat (device) yang ditentukan sebelumnya. Pertama, menggunakan `model_1.to(device)`, model tersebut dipindahkan ke perangkat yang telah diatur sebelumnya, baik itu "cuda" (GPU) atau "cpu" (CPU). Hal ini memastikan bahwa komputasi yang dilakukan oleh model akan sesuai dengan perangkat yang diinginkan. Selanjutnya, dengan menggunakan `list(model_1.parameters())`, kita mendapatkan daftar parameter-pesos dari model. Informasi ini berguna untuk melihat nilai-nilai awal dari parameter-pesos, terutama setelah model dipindahkan ke perangkat yang sesuai. Keseluruhan, kode tersebut menyusun model regresi linear, memindahkannya ke perangkat yang diinginkan, dan menampilkan parameter-pesos model.

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

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

Kode di atas bertujuan untuk menentukan fungsi loss dan optimizer yang akan digunakan selama proses pelatihan model. Pertama, `nn.L1Loss()` digunakan untuk membuat objek fungsi loss dengan menggunakan L1 Loss, yang juga dikenal sebagai Mean Absolute Error (MAE). Fungsi loss ini akan digunakan untuk mengukur seberapa besar selisih absolut antara prediksi model dan nilai target.

Selanjutnya, `torch.optim.SGD(...)` digunakan untuk menginisialisasi optimizer Stochastic Gradient Descent (SGD). Optimizer ini akan mengoptimalkan parameter-pesos model selama proses pelatihan dengan mengurangi nilai loss. Parameter `params=model_1.parameters()` menunjukkan kepada optimizer bahwa parameter yang akan dioptimalkan adalah parameter-pesos dari model `model_1`. Parameter `lr=0.01` menetapkan tingkat pembelajaran (learning rate) sebagai 0.01, yang mengindikasikan seberapa besar langkah-langkah pembelajaran yang akan diambil selama proses optimisasi.

```python
# 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}")
```

Kode di atas menunjukkan implementasi loop pelatihan (training loop) untuk model regresi linear `model_1` selama 300 epoch menggunakan metode Stochastic Gradient Descent (SGD). Sebelum memulai pelatihan, seed acak ditetapkan dengan `torch.manual_seed(42)` untuk memastikan reproduktibilitas hasil acak. Jumlah epoch yang ditetapkan adalah 300. Selanjutnya, data pelatihan dan pengujian (`X_train`, `X_test`, `y_train`, `y_test`) dipindahkan ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU). Loop pelatihan dimulai dengan iterasi sebanyak jumlah epoch yang ditetapkan. Pada setiap iterasi, model diatur dalam mode pelatihan dengan `model_1.train()`.

Proses pelatihan dimulai dengan langkah-langkah berikut:

1. Forward pass: Model melakukan prediksi terhadap data pelatihan (`X_train`).
2. Menghitung loss: Fungsi loss L1 (Mean Absolute Error) digunakan untuk mengukur selisih absolut antara prediksi dan nilai target (`y_train`).

3. Reset gradients: Gradients dari parameter-pesos model direset menjadi nol dengan `optimizer.zero_grad()`.
4. Backpropagation: Gradients loss dihitung kembali melalui model dengan `loss.backward()`.
5. Step optimizer: Optimizer (SGD) melakukan langkah pembelajaran untuk memperbarui nilai parameter-pesos model dengan `optimizer.step()`.

Setiap 20 epoch, dilakukan evaluasi pada data pengujian untuk mengukur performa model di luar data pelatihan. Model diubah menjadi mode evaluasi (`model_1.eval()`) dan dengan menggunakan `torch.inference_mode()`, prediksi model dibuat pada data pengujian (`X_test`) dan dihitung loss pada data pengujian menggunakan fungsi loss L1 (`loss_fn`). Informasi tentang loss pada setiap 20 epoch dicetak untuk melihat perkembangan pelatihan.

Secara keseluruhan, loop pelatihan ini memberikan pemahaman tentang proses pelatihan model regresi linear dengan PyTorch, termasuk proses forward pass, perhitungan loss, backpropagation, dan optimisasi parameter-pesos dengan SGD.

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

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

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

Kode di atas menunjukkan langkah-langkah untuk membuat prediksi dengan model regresi linear (`model_1`) setelah proses pelatihan. Dengan menggunakan `model_1.eval()`, model diubah ke mode evaluasi, yang memastikan bahwa selama inferensi, model tidak melakukan perhitungan gradients, sehingga menghemat sumber daya dan meningkatkan kecepatan eksekusi. Selanjutnya, dengan `with torch.inference_mode():`, sebuah blok konteks diterapkan di mana operasi-inferensi dilakukan. Pada blok ini, prediksi model (`y_preds`) dihasilkan dengan memberikan data pengujian (`X_test`) sebagai input ke model. Hasil prediksi ini memberikan nilai prediksi untuk setiap sampel dalam data pengujian.

Secara keseluruhan, kode ini memberikan cara untuk melakukan inferensi menggunakan model regresi linear yang telah dilatih, dan hasil prediksi (`y_preds`) dapat digunakan untuk evaluasi performa model pada data pengujian.

```
y_preds.cpu()
```

Kode di atas menggunakan metode .cpu() untuk memindahkan hasil prediksi (y_preds) dari tensor yang mungkin berada di GPU ke CPU. Saat model dilatih di GPU, prediksi model juga

cenderung berada di GPU. Namun, dalam beberapa situasi, kita mungkin perlu memindahkan hasil prediksi ke CPU, misalnya untuk keperluan visualisasi atau analisis lebih lanjut di lingkungan CPU.

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

Kode di atas menggunakan fungsi plot_predictions untuk menghasilkan visualisasi prediksi model pada data pengujian. Dengan memberikan argumen predictions=y_preds.cpu(), kita menyertakan hasil prediksi yang telah dipindahkan ke CPU untuk diproses dalam grafik. Fungsi ini akan membuat scatter plot yang menunjukkan titik-titik data pengujian, titik-titik prediksi model, dan titik-titik data pelatihan, jika ada.

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

```
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)
```

Kode di atas berfokus pada penyimpanan model regresi linear yang telah dilatih ke dalam file. Langkah-langkah yang diambil untuk mencapai ini adalah sebagai berikut:

1. Membuat Direktori Model: Pertama, dengan menggunakan `Path("models")`, sebuah obyek `Path` dibuat untuk merepresentasikan direktori yang akan berisi model. Dengan memanggil `.mkdir(parents=True, exist_ok=True)`, kita memastikan bahwa direktori tersebut sudah ada atau akan dibuat jika belum ada.
2. Menentukan Path untuk Penyimpanan Model: Nama model (`MODEL_NAME`) dan path lengkap untuk menyimpan model (`MODEL_SAVE_PATH`) ditentukan. Ini menciptakan path ke file tempat model akan disimpan di dalam direktori models.
3. Menyimpan State Dictionary Model: Dengan menggunakan `torch.save()`, state dictionary dari model (`model_1.state_dict()`) disimpan ke path yang telah ditentukan (`MODEL_SAVE_PATH`). State dictionary ini berisi nilai-nilai dari semua parameter-pesos model dan dapat digunakan untuk memulihkan model di sesi penggunaan berikutnya.

Hasilnya, model regresi linear yang telah dilatih disimpan dalam sebuah file dengan menggunakan state dictionary, memungkinkan untuk penggunaan atau evaluasi lebih lanjut di masa depan tanpa perlu menjalankan pelatihan model kembali.

```
# 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)
```

Kode di atas bertujuan untuk membuat instansi baru dari model regresi linear (`loaded_model`), dan memuat state dictionary yang telah disimpan sebelumnya ke dalam model tersebut. Pertama, menggunakan `LinearRegressionModel()`, sebuah instansi baru dari model regresi linear dibuat dengan nama `loaded_model`. Ini menciptakan model dengan arsitektur yang sama seperti model yang telah dilatih sebelumnya. Selanjutnya, dengan menggunakan `torch.load(f=MODEL_SAVE_PATH)`, state dictionary model yang telah disimpan sebelumnya di-load ke dalam model baru (`loaded_model`). State dictionary ini mengandung nilai-nilai parameter-pesos yang diperlukan untuk mengembalikan model ke keadaan yang sama seperti saat penyimpanan.

Terakhir, menggunakan `loaded_model.to(device)`, model diatur untuk dieksekusi pada perangkat yang telah ditentukan sebelumnya (GPU atau CPU). Ini penting untuk memastikan bahwa model yang telah dimuat sesuai dengan perangkat yang digunakan untuk pelatihan atau penyimpanan sebelumnya. Dengan demikian, model yang telah dimuat siap digunakan untuk evaluasi atau inferensi pada perangkat yang sesuai.

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

Kode di atas bertujuan untuk membuat prediksi menggunakan model yang telah dimuat (`loaded_model`) pada data pengujian (`X_test`) dan membandingkannya dengan prediksi yang telah dilakukan sebelumnya (`y_preds`). Pertama, dengan menggunakan `loaded_model(X_test)`, prediksi model baru dihasilkan untuk data pengujian yang sama. Selanjutnya, menggunakan operator perbandingan `==`, kita membandingkan elemen-elemen prediksi model sebelumnya (`y_preds`) dengan prediksi model yang baru (`y_preds_new`). Hasilnya adalah sebuah tensor boolean yang menunjukkan kesamaan atau ketidaksesamaan antara prediksi model sebelumnya dan prediksi model yang baru untuk setiap sampel dalam data pengujian.

Hal ini memberikan informasi tentang sejauh mana model yang telah dimuat mampu mereproduksi prediksi yang sama dengan model sebelumnya pada data pengujian yang sama. Jika hasilnya adalah tensor boolean yang seluruhnya `True`, maka prediksi kedua model sepenuhnya identik pada data pengujian. Sebaliknya, jika terdapat elemen yang bernilai `False`, maka terdapat perbedaan antara prediksi kedua model untuk sampel tersebut.

```
loaded_model.state_dict()
```

Kode di atas menggunakan metode .state_dict() pada model yang telah dimuat (loaded_model) untuk mengakses dan mencetak state dictionary. State dictionary ini adalah sebuah koleksi yang berisi nilai-nilai dari semua parameter-pesos model, termasuk nilai-nilai yang telah disimpan sebelumnya selama pelatihan.