Nama : Rizka Rahmadina

NIM : 1103204115

Technical Report :

Reproduce_Codes_01_pytorch_workflow

```
import datetime
import torch
import matplotlib.pyplot as plt
from torch import nn
```

- import datetime: Mengimpor modul datetime dari Python, yang menyediakan fungsi-fungsi untuk bekerja dengan tanggal dan waktu.
- import torch: Mengimpor framework PyTorch, sebuah framework machine learning/populasi untuk Python yang digunakan untuk pengolahan tensor dan pembelajaran mesin.
- import matplotlib.pyplot as plt: Mengimpor modul pyplot dari pustaka matplotlib. Ini digunakan untuk membuat visualisasi grafik dan plot data.
- from torch import nn: Mengimpor modul nn (neural network) dari PyTorch. Modul ini menyediakan kelas-kelas yang digunakan untuk membangun dan melatih jaringan saraf.

Kode tersebut merupakan bagian dari pengaturan awal untuk menggunakan PyTorch dan library terkait yang diperlukan untuk penggunaan PyTorch dalam konteks tugas pembelajaran mesin.

```
# Display the last modification date for documentation purposes
last_updated = datetime.datetime.now()
print(f"Document last modified: {last_updated}")
```

- last_updated = datetime.datetime.now(): Membuat objek datetime menggunakan modul datetime yang telah diimpor sebelumnya. datetime.now() memberikan tanggal dan waktu saat ini. Objek datetime ini disimpan dalam variabel last_updated.

- print(f"Document last modified: {last_updated}"): Menggunakan fungsi print untuk mencetak teks ke konsol. Fungsinya mencetak pesan yang berisi informasi tentang terakhir kali dokumen dimodifikasi. Menggunakan f-string untuk memasukkan nilai variabel last_updated ke dalam string.

Kode tersebut berfungsi untuk menampilkan tanggal dan waktu terakhir modifikasi, mencatat waktu terakhir modifikasi dokumen dan menampilkannya dalam bentuk teks di konsol. Hal ini berguna untuk tujuan dokumentasi atau pelacakan waktu terakhir dokumen diubah.

```
# Configure code to be device-agnostic
import torch

# Check if CUDA is available, otherwise use CPU
preferred_device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Selected device: {preferred_device}")
```

- import torch: Mengimpor framework PyTorch, yang akan digunakan untuk operasi-operasi tensor dan pembelajaran mesin.
- preferred_device = "cuda" if torch.cuda.is_available() else "cpu": Baris ini menentukan perangkat yang akan digunakan untuk eksekusi kode. Jika CUDA (GPU acceleration) tersedia (diakses melalui PyTorch), maka perangkat yang dipilih adalah "cuda" (GPU); jika tidak, perangkat yang dipilih adalah "cpu" (Central Processing Unit).
- print(f"Selected device: {preferred_device}"): Mencetak informasi tentang perangkat yang telah dipilih (GPU atau CPU) ke konsol menggunakan f-string.

Kode tersebut berfungsi untuk mengkonfigurasi perangkat (device) yang akan digunakan dalam eksekusi kode, untuk memilih perangkat eksekusi (GPU jika tersedia, jika tidak, maka CPU) secara otomatis. Hal ini berguna ketika Anda ingin menulis kode yang dapat dijalankan pada berbagai perangkat tanpa harus mengganti konfigurasi secara manual. Misalnya, jika GPU tersedia, maka kode akan menggunakan GPU; jika tidak, maka akan menggunakan CPU

```
# Define custom data parameters
custom_weight = 0.5
custom_bias = 1.2

# Generate X and y using different linear regression features
custom_X = torch.arange(0, 1, 0.02).unsqueeze(dim=0).t()
custom_y = custom_weight * custom_X + custom_bias

# Display information about the custom dataset
print(f"Number of X samples: {len(custom_X)}")
print(f"Number of y samples: {len(custom_y)}")
print("First 10 X & y samples:")
print(f"X: {custom_X[:10]}\ny: {custom_y[:10]}")
```

- custom_weight = 0.5: Mendefinisikan bobot (weight) kustom untuk model linear regression.
- custom_bias = 1.2: Mendefinisikan bias (intercept) kustom untuk model linear regression.
- custom_X = torch.arange(0, 1, 0.02).unsqueeze(dim=0).t(): Membuat tensor custom_X yang berisi nilai-nilai dari 0 hingga 1 dengan selang 0.02, dan mengubah dimensinya agar sesuai dengan input model. .unsqueeze(dim=0).t() berfungsi untuk menambah dimensi dan mentransposisi tensor.
- custom_y = custom_weight * custom_X + custom_bias: Menggunakan parameter bobot dan bias yang telah didefinisikan sebelumnya untuk menghasilkan tensor custom_y sesuai dengan model linear regression.
- print(f"Number of X samples: {len(custom_X)}"): Mencetak jumlah sampel dalam tensor custom_X.
- print(f"Number of y samples: {len(custom_y)}"): Mencetak jumlah sampel dalam tensor custom_y.
- print("First 10 X & y samples:"): Mencetak pesan yang menunjukkan bahwa yang akan dicetak selanjutnya adalah 10 sampel pertama dari custom_X dan custom_y.
- print(f"X: {custom_X[:10]}\ny: {custom_y[:10]}"): Mencetak 10 sampel pertama dari custom_X dan custom_y.

Kode tersebut bertujuan untuk mendefinisikan parameter data kustom dan menghasilkan dataset linear regression sederhana, menghasilkan dan menampilkan dataset kustom untuk digunakan dalam suatu model linear regression. Dataset ini

terdiri dari pasangan input-output (X dan y) yang dihasilkan menggunakan parameter bobot dan bias tertentu.

```
# Customized splitting of the custom data into training and testing
custom_train_split = int(len(custom_X) * 0.7)
custom_X_train = custom_X[:custom_train_split]
custom_y_train = custom_y[:custom_train_split]
custom_X_test = custom_X[custom_train_split:]
custom_y_test = custom_y[custom_train_split:]
print(f"Number of X_train samples: {len(custom_X_train)}")
print(f"Number of y_train samples: {len(custom_y_train)}")
print(f"Number of X_test samples: {len(custom_X_test)}")
print(f"Number of y_test samples: {len(custom_y_test)}")
```

- custom_train_split = int(len(custom_X) * 0.7): Menentukan titik pemisahan data antara data latih dan data uji. Dalam hal ini, 70% dari data akan digunakan sebagai data latih (custom_X_train dan custom_y_train), sedangkan 30% sisanya akan digunakan sebagai data uji (custom_X_test dan custom_y_test).
- custom_X_train = custom_X[:custom_train_split]: Memisahkan 70% pertama dari custom_X untuk data latih.
- custom_y_train = custom_y[:custom_train_split]: Memisahkan 70% pertama dari custom_y untuk data latih.
- custom_X_test = custom_X[custom_train_split:]: Memisahkan 30% sisanya dari custom_X untuk data uji.
- custom_y_test = custom_y[custom_train_split:]: Memisahkan 30% sisanya dari custom_y untuk data uji.
- Mencetak informasi tentang jumlah sampel dalam masing-masing dataset (custom_X_train, custom_y_train, custom_X_test, dan custom_y_test).

Kode tersebut bertujuan untuk melakukan pemisahan data kustom menjadi data latih (training) dan data uji (testing), melakukan pemisahan dataset kustom menjadi data latih dan data uji sesuai dengan proporsi yang telah ditentukan (70% latih, 30% uji). Hal ini umumnya dilakukan untuk mengevaluasi kinerja model pada data yang tidak pernah dilihat selama pelatihan.

```
# Customized plot function with different parameters
def custom_plot_predictions(train_data=custom_X_train,
                            train_labels=custom_y_train,
                            test_data=custom_X_test,
                            test_labels=custom_y_test,
                            custom_predictions=None):
    plt.figure(figsize=(12, 8))  # Different figsize
    plt.scatter(train_data, train_labels, c='blue', s=6, label="Custom
Training data")  # Different s (size)
    plt.scatter(test_data, test_labels, c='green', s=6, label="Custom Test
data")  # Different s (size)

    if custom_predictions is not None:
        plt.scatter(test_data, custom_predictions, c='red', s=6,
label="Custom Predictions")  # Different s (size)
    plt.legend(prop={"size": 16})  # Different prop size

# Call the custom plot function
custom_plot_predictions()
```

- def custom_plot_predictions(train_data=custom_X_train, train_labels=custom_y_train, test_data=custom_X_test, test_labels=custom_y_test, custom_predictions=None):: Mendefinisikan fungsi custom_plot_predictions dengan parameter opsional yang dapat disesuaikan (data latih, label latih, data uji, label uji, dan prediksi kustom).
- plt.figure(figsize=(12, 8)): Mengatur ukuran figur plot dengan panjang 12 dan lebar 8.
- plt.scatter(train_data, train_labels, c='blue', s=6, label="Custom Training data"): Membuat scatter plot untuk data latih dengan warna biru, ukuran 6, dan label "Custom Training data".
- plt.scatter(test_data, test_labels, c='green', s=6, label="Custom Test data"): Membuat scatter plot untuk data uji dengan warna hijau, ukuran 6, dan label "Custom Test data".
- if custom_predictions is not None:: Mengecek apakah parameter custom_predictions tidak kosong (tidak None).
- plt.scatter(test_data, custom_predictions, c='red', s=6, label="Custom Predictions"): Jika prediksi kustom diberikan, membuat scatter plot untuk prediksi dengan warna merah, ukuran 6, dan label "Custom Predictions".

- plt.legend(prop={"size": 16}): Menambahkan legenda pada plot dengan ukuran teks 16.
- Memanggil fungsi custom_plot_predictions untuk membuat plot dengan parameter default (menggunakan data latih dan uji yang telah didefinisikan sebelumnya).

Kode ini mendefinisikan sebuah fungsi custom_plot_predictions yang digunakan untuk membuat plot dari data latih, data uji, dan prediksi kustom, untuk mendefinisikan fungsi custom_plot_predictions yang dapat menghasilkan plot dengan parameter-parameter tertentu, seperti data latih, data uji, dan prediksi kustom. Plot ini dapat digunakan untuk memvisualisasikan hasil dari model regresi linear atau tugas prediksi lainnya.

```
# Custom PyTorch linear regression model by subclassing nn.Module
class CustomLinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.custom_weight = nn.Parameter(data=torch.randn(1,
requires_grad=True, dtype=torch.float))
        self.custom_bias = nn.Parameter(data=torch.randn(1,
requires_grad=True, dtype=torch.float))

    def forward(self, x):
        return self.custom_weight * x + self.custom_bias

# Instantiate the custom model
torch.manual_seed(24)
custom_model = CustomLinearRegressionModel()
custom_model, custom_model.state_dict()
```

- class CustomLinearRegressionModel(nn.Module):: Mendefinisikan kelas CustomLinearRegressionModel yang merupakan turunan dari kelas nn.Module dalam PyTorch. Ini adalah pendekatan umum untuk mendefinisikan model di PyTorch.
- def __init__(self):: Metode konstruktor (__init__) dari kelas ini. Dipanggil saat objek model dibuat. Di dalamnya, kita mendefinisikan parameter-parameter model, yaitu bobot (custom_weight) dan bias (custom_bias) sebagai objek nn.Parameter. Objek nn.Parameter digunakan untuk

menandakan bahwa parameter tersebut dapat diubah selama proses pelatihan (requires_grad=True), dan kita memberikan nilai awal dari distribusi normal (torch.randn).

- def forward(self, x):: Metode forward yang mendefinisikan bagaimana input x akan diproses oleh model. Dalam hal ini, model melakukan operasi regresi linear dengan bobot (custom_weight) dan bias (custom_bias).
- torch.manual_seed(24): Menetapkan seed untuk mengontrol randomness dan memastikan reproduktibilitas hasil ketika menggunakan operasi-acak di PyTorch.
- custom_model = CustomLinearRegressionModel(): Menginstansiasi objek model dari kelas CustomLinearRegressionModel.
- custom_model.state_dict(): Mengambil state dictionary dari model. State dictionary adalah koleksi dari semua parameter yang ada dalam model beserta nilai-nilai mereka.

Kode tersebut mendefinisikan sebuah model regresi linear kustom menggunakan PyTorch dengan cara menurunkan kelas nn.Module, mendefinisikan model regresi linear kustom dengan menggunakan PyTorch. Model ini memiliki dua parameter yang dapat diubah, yaitu bobot (custom_weight) dan bias (custom_bias). Parameter-parameter ini akan disesuaikan selama proses pelatihan untuk meminimalkan fungsi kerugian dan meningkatkan kinerja model. Model juga telah diinisialisasi dengan seed tertentu untuk menjamin reproduktibilitas hasil.

```
# Check the device of the first parameter in the custom model
next(custom_model.parameters()).device
```

- next(custom_model.parameters()).device: Mencoba untuk mendapatkan perangkat (device) dari parameter pertama dalam model kustom. custom_model.parameters() mengembalikan iterator yang memberikan semua parameter dalam model, dan next() digunakan untuk mendapatkan elemen pertama dari iterator tersebut. Kemudian, .device digunakan untuk mendapatkan perangkat pada which parameter tersebut berada.

Kode tersebut bertujuan untuk memeriksa perangkat (device) tempat parameter pertama dalam model kustom (custom_model) berada, untuk memeriksa perangkat di mana parameter pertama dari model kustom berada. Hal ini dapat berguna untuk

memastikan bahwa model dan parameter-parameternya berada pada perangkat yang diinginkan (seperti GPU atau CPU).

```
# Instantiate the custom model and move it to the preferred device
custom_model.to(preferred_device)
list(custom_model.parameters())
```

- custom_model.to(preferred_device): Memindahkan model kustom (custom_model) ke perangkat yang dipilih sebelumnya (preferred_device). Fungsi to() digunakan untuk memindahkan model dan parameter-parameternya ke perangkat yang diinginkan.
- list(custom_model.parameters()): Mengambil daftar parameter dari model kustom. Ini dapat digunakan untuk memeriksa apakah model sekarang berada pada perangkat yang diinginkan dan apakah pemindahan perangkat berjalan dengan sukses.

Inti dari kode ini adalah untuk menginstansiasi model kustom dan memindahkannya ke perangkat yang telah dipilih sebelumnya (preferred_device). Hal ini memastikan bahwa model dan semua parameter-parameternya berada pada perangkat yang diinginkan untuk pelatihan dan evaluasi.

```
# Define the loss function and optimizer for the custom model
custom_loss_fn = nn.L1Loss()
custom_optimizer = torch.optim.SGD(params=custom_model.parameters(),
lr=0.005)
```

- custom_loss_fn = nn.L1Loss(): Mendefinisikan fungsi kerugian dengan menggunakan nn.L1Loss(). Fungsi ini merupakan loss function yang menghitung mean absolute error (MAE) antara prediksi dan target. Fungsi L1Loss sering digunakan untuk tugas regresi.
- custom_optimizer = torch.optim.SGD(params=custom_model.parameters(), lr=0.005): Mendefinisikan pengoptimal dengan menggunakan Stochastic Gradient Descent (SGD). custom_model.parameters() memberikan semua parameter dari model kustom sebagai parameter yang akan dioptimalkan.

lr=0.005 adalah tingkat pembelajaran (learning rate) yang digunakan oleh optimizer untuk mengatur seberapa besar langkah yang diambil dalam arah yang mengurangi loss.

Kode tersebut bertujuan untuk mendefinisikan fungsi kerugian (loss function) dan pengoptimal (optimizer) untuk model kustom (custom_model), untuk menetapkan fungsi kerugian dan optimizer yang akan digunakan selama pelatihan model kustom. Dalam hal ini, MAE (L1 loss) digunakan sebagai fungsi kerugian, dan SGD digunakan sebagai optimizer dengan learning rate sebesar 0.005.

```
# Custom training loop for the custom model for 300 epochs
torch.manual_seed(24)  # Different seed for variety

custom_epochs = 300

# Send data to the target device
custom_X_train = custom_X_train.to(preferred_device)
custom_X_test = custom_X_test.to(preferred_device)
custom_y_train = custom_y_train.to(preferred_device)
custom_y_test = custom_y_test.to(preferred_device)

for custom_epoch in range(custom_epochs):
    ### Training

    # Put the custom model in train mode
    custom_model.train()

    # 1. Forward pass
    custom_y_pred = custom_model(custom_X_train)

    # 2. Calculate loss
    custom_loss = custom_loss_fn(custom_y_pred, custom_y_train)

    # 3. Zero gradients
    custom_optimizer.zero_grad()

    # 4. Backpropagation
    custom_loss.backward()

    # 5. Step the optimizer
    custom_optimizer.step()
```

```
    ### Perform testing every 20 epochs
    if custom_epoch % 15 == 0:
        # Put the custom model in evaluation mode and setup inference
context
        custom_model.eval()
        with torch.no_grad():
            # 1. Forward pass
            custom_y_preds = custom_model(custom_X_test)
            # 2. Calculate test loss
            custom_test_loss = custom_loss_fn(custom_y_preds,
custom_y_test)
            # Print out what's happening
            print(f"Epoch: {custom_epoch} | Train loss: {custom_loss:.2f}
| Test loss: {custom_test_loss:.2f}")
```

- torch.manual_seed(24): Mengatur seed untuk mengontrol randomness dan memastikan reproduktibilitas hasil. Seed yang berbeda digunakan untuk memberikan variasi.
- custom_epochs = 300: Menetapkan jumlah epoch (iterasi pelatihan) yang akan dilakukan, dalam hal ini sebanyak 300 epoch.
- Memindahkan data latih dan uji ke perangkat yang diinginkan (preferred_device). Hal ini diperlukan karena model dan parameter-parameternya juga telah dipindahkan ke perangkat yang sama sebelumnya.
- custom_model.train(): Menetapkan model ke mode pelatihan. Ini diperlukan karena beberapa lapisan atau modul di PyTorch memiliki perilaku yang berbeda antara mode pelatihan dan evaluasi (misalnya, dropout).
- custom_model(custom_X_train): Melakukan langkah forward pass untuk menghasilkan prediksi menggunakan model kustom terhadap data latih.
- custom_loss_fn(custom_y_pred, custom_y_train): Menghitung nilai loss menggunakan fungsi kerugian yang telah didefinisikan sebelumnya (custom_loss_fn).
- custom_optimizer.zero_grad(): Mengatur semua gradien parameter ke nol sebelum melakukan backpropagation pada langkah selanjutnya.
- custom_loss.backward(): Melakukan backpropagation untuk menghitung gradien dari loss terhadap parameter.
- custom_optimizer.step(): Melakukan langkah optimasi untuk memperbarui nilai parameter berdasarkan gradien yang dihitung sebelumnya.

- Setelah setiap 15 epoch (diatur dengan if custom_epoch % 15 == 0:), model dipindahkan ke mode evaluasi (custom_model.eval()). Ini diperlukan karena beberapa modul di PyTorch, seperti dropout, berperilaku berbeda antara mode pelatihan dan evaluasi.
- Dengan menggunakan torch.no_grad(), mengindikasikan bahwa komputasi di dalam blok tersebut tidak perlu melacak gradien, karena ini adalah evaluasi.
- custom_model(custom_X_test): Melakukan langkah forward pass untuk menghasilkan prediksi menggunakan model kustom terhadap data uji.
- custom_loss_fn(custom_y_preds, custom_y_test): Menghitung nilai loss pada data uji.
- Mencetak informasi tentang epoch, nilai loss pada data latih (custom_loss), dan nilai loss pada data uji (custom_test_loss). Informasi ini dapat digunakan untuk memantau kinerja model selama pelatihan.

Kode tersebut merupakan sebuah loop pelatihan kustom untuk model yang telah didefinisikan sebelumnya (custom_model). Loop ini dilakukan selama 300 epoch (iterasi pelatihan).

```
# Make predictions with the custom model
custom_model.eval()

with torch.no_grad():
    custom_y_preds = custom_model(custom_X_test)
custom_y_preds
```

- custom_model.eval(): Mengatur model ke mode evaluasi. Ini diperlukan karena beberapa modul di PyTorch, seperti dropout, berperilaku berbeda antara mode pelatihan dan evaluasi.
- with torch.no_grad():: Mengindikasikan bahwa komputasi di dalam blok tersebut tidak perlu melacak gradien. Hal ini diperlukan untuk membuat prediksi tanpa mempengaruhi nilai-nilai gradien dalam model.
- custom_y_preds = custom_model(custom_X_test): Melakukan langkah forward pass untuk menghasilkan prediksi menggunakan model kustom terhadap data uji (custom_X_test). Hasil prediksi disimpan dalam variabel custom_y_preds.

- Mencetak hasil prediksi. Hasil ini dapat digunakan untuk mengevaluasi sejauh mana model kustom telah belajar memprediksi data yang belum pernah dilihatnya sebelumnya (data uji).

Kode ini bertujuan untuk membuat prediksi menggunakan model kustom (custom_model) pada data uji (custom_X_test).

```
# Move predictions to CPU for compatibility with non-CUDA-enabled
libraries
custom_y_preds_cpu = custom_y_preds.cpu()
```

Kode ini bertujuan untuk memindahkan hasil prediksi (custom_y_preds) dari perangkat GPU (jika model berada di GPU) ke CPU. Hal ini dilakukan untuk memastikan kompatibilitas dengan library atau operasi lain yang tidak mendukung CUDA (GPU acceleration).

- custom_y_preds.cpu(): Memindahkan tensor custom_y_preds dari perangkat (device) GPU ke CPU menggunakan metode .cpu(). Jika model dan data uji berada di GPU, ada kemungkinan bahwa operasi selanjutnya tidak mendukung CUDA, sehingga kita perlu memastikan bahwa data berada di CPU.
- Hasilnya disimpan dalam variabel custom_y_preds_cpu. Data ini sekarang berada di CPU dan dapat digunakan dengan library atau operasi yang tidak mendukung CUDA.

```
import matplotlib.pyplot as plt

# Define the plot_predictions function
def plot_predictions(test_data, test_labels, predictions):
    plt.figure(figsize=(10, 6))
    plt.scatter(test_data, test_labels, label='True test data',
color='green')
    plt.scatter(test_data, predictions, label='Predictions', color='red',
marker='x')
    plt.title('Linear Regression: True Test Data and Predictions')
    plt.xlabel('X')
    plt.ylabel('y')
```

```
    plt.legend()
    plt.show()

# Plot the predictions for the custom model (ensure predictions are on
CPU)
plot_predictions(test_data=custom_X_test.cpu(),
                 test_labels=custom_y_test.cpu(),
                 predictions=custom_y_preds_cpu)
```

Kode ini mendefinisikan sebuah fungsi plot_predictions dan menggunakan fungsi tersebut untuk membuat plot hasil prediksi model terhadap data uji.

- Mengimpor library matplotlib.pyplot untuk membuat plot.
- Mendefinisikan fungsi plot_predictions yang akan digunakan untuk membuat plot hasil prediksi model. Fungsi ini menerima tiga parameter: test_data (data uji), test_labels (label sebenarnya dari data uji), dan predictions (hasil prediksi model).
- plt.figure(figsize=(10, 6)): Membuat sebuah figur dengan ukuran 10x6.
- plt.scatter(test_data, test_labels, label='True test data', color='green'): Membuat scatter plot untuk menampilkan data uji sebenarnya (label hijau).
- plt.scatter(test_data, predictions, label='Predictions', color='red', marker='x'): Membuat scatter plot untuk menampilkan hasil prediksi model (label merah dengan tanda 'x').
- plt.title('Linear Regression: True Test Data and Predictions'): Menambahkan judul pada plot.
- plt.xlabel('X') dan plt.ylabel('y'): Menambahkan label sumbu x dan y pada plot.
- plt.legend(): Menambahkan legenda ke plot untuk membedakan antara data uji sebenarnya dan hasil prediksi.
- plt.show(): Menampilkan plot.
- Memanggil fungsi plot_predictions dengan memberikan data uji (custom_X_test.cpu()), label sebenarnya (custom_y_test.cpu()), dan hasil prediksi (custom_y_preds_cpu). Data uji dan hasil prediksi harus dipindahkan ke CPU untuk memastikan kompatibilitas dengan library matplotlib yang mungkin tidak mendukung CUDA.

```
from pathlib import Path

# 1. Create a directory for models
CUSTOM_MODEL_PATH = Path("custom_models")
CUSTOM_MODEL_PATH.mkdir(parents=True, exist_ok=True)

# 2. Create a path for saving the custom model
CUSTOM_MODEL_NAME = "custom_linear_regression_model"
CUSTOM_MODEL_SAVE_PATH = CUSTOM_MODEL_PATH / CUSTOM_MODEL_NAME

# 3. Save the custom model state_dict()
print(f"Saving custom model to {CUSTOM_MODEL_SAVE_PATH}")
torch.save(obj=custom_model.state_dict(), f=CUSTOM_MODEL_SAVE_PATH)
```

Kode tersebut bertujuan untuk menyimpan state dictionary dari model kustom (custom_model) ke dalam file.

- Mengimpor kelas Path dari library pathlib untuk bekerja dengan path file dan direktori.
- CUSTOM_MODEL_PATH = Path("custom_models"): Mendefinisikan path direktori untuk menyimpan model, dalam hal ini, direktori bernama "custom_models".
- CUSTOM_MODEL_PATH.mkdir(parents=True, exist_ok=True): Membuat direktori tersebut. parents=True memastikan bahwa direktori induk (jika tidak ada) juga akan dibuat, dan exist_ok=True memastikan bahwa tidak ada error yang dihasilkan jika direktori tersebut sudah ada.
- CUSTOM_MODEL_NAME = "custom_linear_regression_model": Mendefinisikan nama model, dalam hal ini, "custom_linear_regression_model".
- CUSTOM_MODEL_SAVE_PATH = CUSTOM_MODEL_PATH / CUSTOM_MODEL_NAME: Menggabungkan path direktori (CUSTOM_MODEL_PATH) dengan nama model (CUSTOM_MODEL_NAME) untuk mendapatkan path lengkap tempat model akan disimpan.
- Mencetak informasi bahwa proses penyimpanan model dimulai.
- torch.save(obj=custom_model.state_dict(), f=CUSTOM_MODEL_SAVE_PATH): Menyimpan state dictionary dari model kustom ke file yang telah ditentukan sebelumnya

(CUSTOM_MODEL_SAVE_PATH). State dictionary ini mencakup parameter-parameter yang telah dipelajari oleh model selama proses pelatihan. Model dapat di-beban kembali dari file ini di waktu yang akan datang untuk digunakan atau evaluasi.

```
# Create a new instance of the custom model and load the saved
state_dict() (put it on the target device)
loaded_custom_model = CustomLinearRegressionModel()
loaded_custom_model.load_state_dict(torch.load(f=CUSTOM_MODEL_SAVE_PATH))
loaded_custom_model.to(preferred_device)
```

Kode tersebut bertujuan untuk membuat sebuah instance baru dari model kustom (CustomLinearRegressionModel) dan memuat state dictionary yang telah disimpan sebelumnya ke dalamnya. Selanjutnya, model tersebut dipindahkan ke perangkat (device) yang diinginkan (preferred_device).

- loaded_custom_model = CustomLinearRegressionModel(): Menginstansiasi objek model baru dari kelas CustomLinearRegressionModel. Ini menciptakan model dengan parameter-parameter yang belum dipelajari.
- loaded_custom_model.load_state_dict(torch.load(f=CUSTOM_MODEL_SAVE_PATH)): Memuat state dictionary yang telah disimpan sebelumnya menggunakan fungsi torch.load(). State dictionary ini berisi parameter-parameter yang telah dipelajari oleh model selama pelatihan sebelumnya. Model yang baru diinstansiasi kemudian diperbarui dengan parameter-parameter ini.
- loaded_custom_model.to(preferred_device): Memindahkan model yang baru dimuat ke perangkat yang diinginkan (preferred_device). Ini dilakukan untuk memastikan bahwa model berada pada perangkat yang sesuai dengan kebutuhan (GPU atau CPU).
- Inti dari kode ini adalah untuk membuat model baru, memuat parameter-parameter yang telah disimpan sebelumnya, dan memindahkan model ke perangkat yang diinginkan untuk digunakan atau evaluasi lebih lanjut. Model ini sekarang siap untuk melakukan prediksi atau pelatihan tambahan, jika diperlukan.

```
# Make predictions with the loaded custom model and compare them to the
previous predictions
custom_y_preds_new = loaded_custom_model(custom_X_test)
result_comparison = torch.equal(custom_y_preds, custom_y_preds_new)
print(result_comparison)
```

Kode tersebut bertujuan untuk membuat prediksi menggunakan model kustom yang baru dimuat (loaded_custom_model) dan membandingkannya dengan hasil prediksi sebelumnya (custom_y_preds).

- custom_y_preds_new = loaded_custom_model(custom_X_test): Menggunakan model yang baru dimuat (loaded_custom_model), melakukan langkah forward pass untuk menghasilkan prediksi terhadap data uji yang sama (custom_X_test). Hasil prediksi baru disimpan dalam variabel custom_y_preds_new.
- result_comparison = torch.equal(custom_y_preds, custom_y_preds_new): Membandingkan hasil prediksi yang baru (custom_y_preds_new) dengan hasil prediksi sebelumnya (custom_y_preds). Fungsi torch.equal() mengembalikan True jika kedua tensor memiliki nilai yang sama, dan False jika tidak.
- print(result_comparison): Mencetak hasil perbandingan. Jika hasil prediksi baru dan prediksi sebelumnya sama, output akan menjadi True; sebaliknya, jika berbeda, output akan menjadi False.
- Inti dari kode ini adalah untuk membandingkan hasil prediksi yang baru dibuat dengan model yang baru dimuat dengan hasil prediksi sebelumnya yang telah disimpan sebelumnya. Hal ini membantu memastikan bahwa model yang di-load dapat menghasilkan prediksi yang konsisten dengan model asli. Jika result_comparison adalah True, maka prediksi model yang baru dimuat identik dengan prediksi sebelumnya.

```
# Get the state_dict() of the loaded custom model
state_dict_loaded_model = loaded_custom_model.state_dict()
print(state_dict_loaded_model)
```

- state_dict_loaded_model = loaded_custom_model.state_dict(): Mendapatkan state dictionary dari model yang baru dimuat (loaded_custom_model). State dictionary ini berisi parameter-parameter model seperti bobot dan bias, dan nilainya setelah proses pelatihan sebelumnya.
- print(state_dict_loaded_model): Mencetak state dictionary dari model yang baru dimuat. Ini dapat digunakan untuk melihat nilai parameter-parameter yang ada dalam model kustom tersebut.

Kode ini bertujuan untuk mendapatkan state dictionary dari model kustom yang baru dimuat (loaded_custom_model). State dictionary ini mencakup parameter-parameter yang telah dipelajari oleh model selama pelatihan sebelumnya, untuk mendapatkan dan mencetak state dictionary dari model kustom yang baru dimuat. State dictionary ini dapat digunakan untuk memahami dan memeriksa nilai-nilai parameter yang dimiliki oleh model tersebut setelah proses pelatihan sebelumnya.

Kode-kode diatas merupakan bentuk dari implementasi regresi linear menggunakan PyTorch yang dilakukan pengaturan awal dengan mengimpor library yang diperlukan seperti torch, matplotlib, dan nn (PyTorch's neural network module). Selanjutnya, dilakukan pembuatan model regresi linear kustom dengan menggunakan PyTorch's nn.Module. Parameter-parameter model ini kemudian disimpan dan di-load untuk memastikan konsistensi antara model yang di-training dan model yang di-load.

Data latih dan uji di-generate, lalu di-split menjadi data training dan testing. Kemudian, dilakukan pelatihan model menggunakan custom training loop, di mana dilakukan iterasi sebanyak 300 epoch. Setiap 15 epoch, dilakukan evaluasi model pada data uji dan dicetak nilai loss-nya. Setelah pelatihan, model digunakan untuk membuat prediksi pada data uji.

Model dan hasil prediksi kemudian dipindahkan dari perangkat GPU ke CPU untuk kompatibilitas dengan library yang tidak mendukung CUDA. Plot hasil prediksi pada data uji juga dilakukan untuk memvisualisasikan performa model.

Selanjutnya, model disimpan dalam file untuk digunakan kembali di masa depan. Model tersebut kemudian di-load, dan hasil prediksi yang baru dihasilkan dibandingkan dengan hasil prediksi sebelumnya untuk memastikan konsistensi model, dan state dictionary dari model yang baru dimuat diambil untuk memeriksa nilai-nilai parameter yang dimiliki oleh model tersebut. Keseluruhan kodingan mencakup proses pembuatan model, pelatihan, evaluasi, penyimpanan, dan penggunaan kembali model untuk membuat prediksi.