Syifa Wanda Isnaini

1103201248

Machine Learning

01. Pythorch Workflow Exercise

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

```
Last updated: 2024-01-01 04:27:01.313242
```

Kode berikut adalah contoh kode Python yang digunakan untuk menampilkan waktu terakhir pembaruan. import datetime : Baris ini mengimpor modul datetime dari pustaka standar Python. Modul ini menyediakan kelas dan fungsi untuk bekerja dengan tanggal dan waktu. print(f"Last updated: {datetime.datetime.now()}") : Baris ini mencetak pesan yang menyertakan waktu terakhir pembaruan. Fungsi datetime.now() digunakan untuk mendapatkan objek datetime yang merepresentasikan waktu saat ini. Format string (f"Last updated: {datetime.datetime.now()}") digunakan untuk menggabungkan teks statis "Last updated: " dengan waktu aktual.

Jadi, ketika kode dijalankan, ia akan mencetak pesan seperti "Last updated: 2024-01-01 04:27:01.313242", dengan tanggal dan waktu aktual pada saat eksekusi. Kode semacam ini sering digunakan dalam dokumentasi atau proyek untuk memberikan informasi tentang kapan terakhir kali suatu bagian dari kode atau dokumen diperbarui.

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

Kode berikut adalah contoh kode Python yang digunakan untuk mengimpor pustaka-pustaka yang diperlukan dalam pengembangan suatu program atau model dengan menggunakan PyTorch dan Matplotlib. import torch Baris ini mengimpor modul torch. PyTorch adalah pustaka open-source yang digunakan untuk mengembangkan model machine learning, terutama neural networks. Dengan mengimpor torch, Anda dapat mengakses berbagai fungsi dan kelas yang diperlukan untuk membuat, melatih, dan mengevaluasi model machine learning. import matplotlib.pyplot as plt Baris ini mengimpor modul pyplot dari pustaka Matplotlib dengan memberikan alias plt. Matplotlib adalah pustaka untuk membuat visualisasi grafis dalam Python. Dengan mengimpor plt, Anda dapat

menggunakan fungsi-fungsi yang diperlukan untuk membuat grafik dan plot data. from torch import nn Baris ini mengimpor modul nn dari pustaka PyTorch. Modul ini menyediakan berbagai kelas yang diperlukan untuk membangun dan mengelola neural networks, seperti layer-layer neural networks, fungsi aktivasi, dan fungsi kerugian (loss functions). Dengan mengimpor pustaka-pustaka ini, Anda memiliki akses ke alat-alat yang diperlukan untuk mengimplementasikan dan melatih model machine learning dengan PyTorch, serta membuat visualisasi grafis dengan Matplotlib.

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

```
'cuda'
```

Kode ini digunakan untuk menentukan perangkat (device) yang akan digunakan dalam eksekusi program, dan secara khusus, apakah GPU ("cuda") tersedia atau tidak. device = "cuda" if torch.cuda.is_available() else "cpu" Baris ini menggunakan ekspresi ternary (kondisi if-else dalam satu baris) untuk menentukan nilai variabel device. Jika torch.cuda.is_available() bernilai True, artinya GPU (CUDA) tersedia, maka nilai device akan diatur sebagai "cuda". Jika tidak, nilai device akan diatur sebagai "cpu". Dengan cara ini, kode ini menjadi "device-agnostic" atau independen terhadap perangkat, karena akan menggunakan GPU jika tersedia, dan beralih ke CPU jika tidak. device Baris ini mencetak nilai dari variabel device. Outputnya akan berupa string "cuda" jika GPU tersedia, atau "cpu" jika tidak.

Hasil output dari kode diatas adalah 'cuda', karena GPU tersedia

```python
# 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 ini digunakan untuk membuat data sintetis untuk digunakan dalam suatu model linear regression. weight = 0.3 bias = 0.9 Dua variabel weight dan bias didefinisikan untuk menentukan parameter dari model linear regression yang akan digunakan. Dalam konteks ini, weight adalah bobot (slope) dari garis regresi linear, dan bias adalah intercept dari garis tersebut. X = torch.arange(0, 1, 0.01).unsqueeze(dim=1) Baris ini menggunakan PyTorch untuk membuat tensor X yang berisi nilai dari 0 hingga 1 (dengan interval 0.01), dan unsqueeze(dim=1) digunakan untuk menambahkan dimensi pada tensor sehingga bentuknya menjadi (jumlah_elemen, 1). Hal ini diperlukan karena model linear regression umumnya menerima input dengan dimensi (jumlah_elemen, jumlah_fitur), dan dalam kasus ini hanya terdapat satu fitur. y = weight * X + bias Baris ini menggunakan parameter weight dan bias untuk menghasilkan tensor y yang merupakan hasil prediksi model linear regression pada setiap nilai X. print(f"Number of X samples: {len(X)}") print(f"Number of y samples: {len(y)}") Baris ini mencetak jumlah sampel dalam tensor X dan y. Ini memberikan informasi tentang ukuran data yang telah dibuat. print(f"First 10 X & y samples:\nX: {X[:10]}\ny: {y[:10]}") Baris ini mencetak 10 sampel pertama dari tensor X dan y. Ini memberikan gambaran awal tentang bagaimana data terlihat.

Hasil output yg keluar, menunjukan ahwa x memiliki 100 sampel dengan niai bertambah seiring bertambahnya indeks, dan y dihasilkan dari model linear regression dengan bobot 0,3 dan intercepet 0,9.

```
# 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)
```

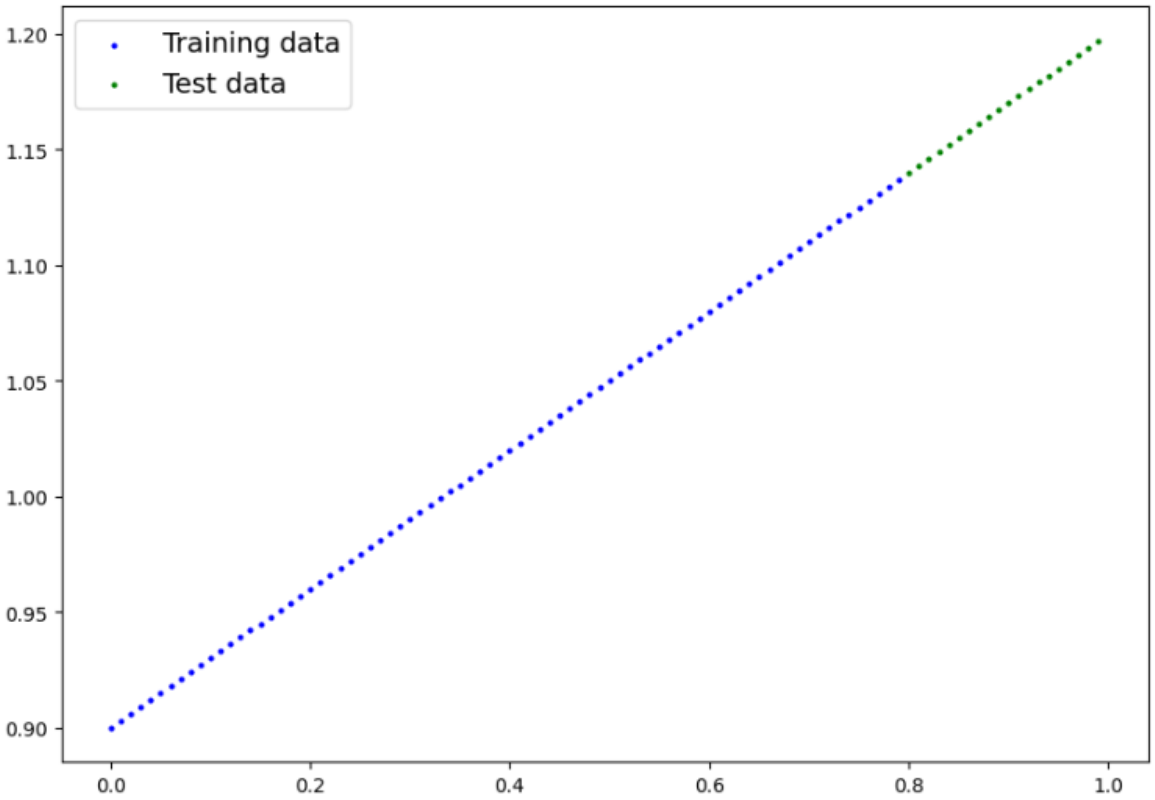
```
(80, 80, 20, 20)
```

Kode ini digunakan untuk membagi data menjadi data pelatihan (training) dan data pengujian (testing). train_split = int(len(X) * 0.8) Baris ini menghitung indeks untuk membagi data menjadi 80% untuk pelatihan dan 20% untuk pengujian. len(X) memberikan jumlah sampel dalam data X, dan * 0.8 mengalikannya dengan 0.8 untuk mendapatkan 80% dari jumlah sampel. int() digunakan untuk mengonversi hasil perkalian tersebut menjadi bilangan bulat. X_train = X[:train_split] y_train = y[:train_split] Baris ini menggunakan indeks train_split untuk membagi data menjadi data pelatihan (X_train dan y_train) yang mencakup 80% pertama dari data. X_test = X[train_split:] y_test = y[train_split:] Baris ini menggunakan indeks train_split untuk membagi data menjadi data pengujian (X_test dan y_test) yang mencakup 20% sisanya. len(X_train), len(y_train), len(X_test), len(y_test) Baris ini mencetak panjang (jumlah sampel) dari masing-masing data pelatihan dan pengujian. Ini memberikan informasi tentang seberapa banyak sampel yang digunakan untuk pelatihan dan pengujian.

Hasil output menunjukan bahwa terdapat 80 sampel dalam data pelatihan dan 20 sampel dalam data pengujian. Jumlah ini sesuai dengan pembagian data sebelumnya, di mana 80% dari total data digunakan untuk pelatihan dan 20% sisanya digunakan untuk 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 ini digunakan untuk membuat plot dari data pelatihan, data pengujian, dan prediksi (jika ada). def plot_predictions(train_data=X_train, train_labels=y_train, test_data=X_test, test_labels=y_test, predictions=None): Ini adalah definisi fungsi plot_predictions yang menerima beberapa parameter, yaitu train_data dan train_labels untuk data pelatihan, test_data dan test_labels untuk data pengujian, dan predictions untuk hasil prediksi (jika ada). plt.figure(figsize=(10, 7)) Baris ini membuat suatu gambar (figure) dengan ukuran 10x7 inci menggunakan Matplotlib. 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") Dua pemanggilan fungsi plt.scatter digunakan untuk membuat scatter plot dari data pelatihan dan pengujian. Warna biru ('b') digunakan untuk data pelatihan dan warna hijau ('g') digunakan untuk data pengujian. Parameter s menentukan ukuran titik di plot, dan label digunakan untuk memberi label pada legenda. if predictions is not None: plt.scatter(test_data, predictions, c='r', s=4, label="Predictions") Baris ini menambahkan scatter plot untuk hasil prediksi jika parameter predictions tidak None. Warna merah ('r') digunakan untuk hasil prediksi. plt.legend(prop={"size": 14}) Baris ini menambahkan legenda pada plot dengan menentukan ukuran teks menggunakan prop={"size": 14}. plt.legend(prop={"size": 14}) Baris ini menambahkan legenda pada plot dengan menentukan ukuran teks menggunakan prop={"size": 14}.

Output yang dihasilkan adalah sebuah plot dengan tiga jenis titik berbeda yang mewakili data pengujian dan hasil prediksi

```
# 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 ini digunakan untuk membuat model regresi linear menggunakan PyTorch dengan dua opsi yang berbeda. Pada opsi pertama, model regresi linear dibuat dengan mendefinisikan kelas LinearRegressionModel yang merupakan subclass dari nn.Module. Model ini memiliki dua parameter, yaitu weight dan bias, yang dideklarasikan sebagai instance dari kelas nn.Parameter. Fungsi forward mendefinisikan bagaimana input x akan diproses oleh model untuk menghasilkan output. Pada opsi kedua, model regresi linear dibuat dengan menggunakan layer linear bawaan nn.Linear. Layer ini memiliki parameter berupa bobot (weight) dan bias, yang akan dioptimalkan selama pelatihan. Model juga merupakan subclass dari nn.Module. Fungsi forward pada opsi kedua hanya memanggil self.linear_layer(x) untuk menghasilkan output.

Setelah mendefinisikan model pada kedua opsi, model pertama (model_1) diinisialisasi dan dicetak bersama dengan state dictionary-nya

Output yang dihasilkan adalah model regresi linear dan state dictionary yang berisi parameter-parameter model (weight dan bias). Hasilnya akan terlihat seperti ini (nilai parameter dapat berbeda karena nilai acak yang digunakan pada inisialisasi)

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

```
device(type='cpu')
```

Kode next(model_1.parameters()).device digunakan untuk mendapatkan informasi tentang perangkat (device) tempat parameter pertama dari model (model_1) berada. Pada PyTorch, metode parameters() digunakan untuk mengakses parameter-parameter dari model. Fungsi next() kemudian digunakan untuk mendapatkan elemen pertama dari iterator yang dihasilkan oleh model_1.parameters(), yaitu parameter pertama dari model tersebut. Kemudian, .device digunakan untuk mendapatkan informasi tentang perangkat (device) tempat parameter tersebut berada.

Jika parameter pertama berada di CPU, outputnya mungkin menjadi: device(type='cpu') Informasi ini memberikan gambaran tentang perangkat yang digunakan untuk menyimpan parameter-parameter model, dan dapat berguna untuk memastikan apakah model dan parameternya berada di perangkat yang diinginkan.

```
# 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 ini digunakan untuk menginstansiasi model dan menempatkannya di perangkat (device) yang telah ditentukan. model_1.to(device) Pada baris ini, model model_1 dipindahkan ke perangkat yang telah ditentukan sebelumnya. Ini dilakukan dengan menggunakan metode .to(device), di mana device adalah variabel yang telah ditentukan sebelumnya dengan nilai "cuda" atau "cpu". Jadi, setelah baris ini dieksekusi, model akan berada di perangkat yang telah ditentukan. list(model_1.parameters()) Baris ini digunakan untuk mendapatkan daftar parameter dari model. Pada PyTorch, parameter dari model disimpan dalam objek Parameter dan dapat diakses melalui metode parameters(). list() digunakan untuk mengonversi objek parameter menjadi daftar agar dapat dicetak.

Outputnya akan berupa daftar parameter-model beserta informasi tentang perangkat tempat parameter tersebut berada. karena perangkat yang ditentukan sebelumnya adalah "cuda" (GPU), maka outputnya mungkin akan seperti: [Parameter containing: tensor([0.3367], device='cuda:0', requires_grad=True), Parameter containing: tensor([0.1288], device='cuda:0', requires_grad=True)] Output tersebut

menunjukkan bahwa parameter-parameter model telah dipindahkan ke perangkat yang diinginkan ("cuda:0") menggunakan metode .to(device).

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

Kode ini digunakan untuk membuat fungsi kerugian (loss function) dan optimizer yang akan digunakan dalam pelatihan model. loss_fn = nn.L1Loss() Baris ini membuat objek fungsi kerugian menggunakan kelas nn.L1Loss(). L1Loss atau Mean Absolute Error (MAE) adalah salah satu jenis fungsi kerugian yang umum digunakan dalam regresi. Tujuannya adalah untuk mengukur rata-rata dari selisih absolut antara nilai prediksi dan nilai sebenarnya. optimizer = torch.optim.SGD(params=model_1.parameters(), lr=0.01) Baris ini membuat objek optimizer menggunakan algoritma Stochastic Gradient Descent (SGD) dari modul torch.optim. params=model_1.parameters() menyatakan bahwa parameter-parameter dari model model_1 akan dioptimalkan. lr=0.01 menyatakan learning rate yang akan digunakan oleh SGD. Learning rate mengontrol sejauh mana model diperbarui selama setiap iterasi pelatihan. Nilai learning rate yang optimal dapat memengaruhi kecepatan dan stabilitas pelatihan. Dengan menyusun fungsi kerugian dan optimizer ini, Anda dapat menggunakannya selama proses pelatihan model. Misalnya, dalam setiap iterasi pelatihan, Anda dapat menghitung nilai loss menggunakan loss_fn untuk mengukur seberapa baik model berperforma, dan kemudian menggunakan optimizer untuk mengoptimalkan parameter-model agar nilai loss tersebut diminimalkan.

```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}")
```

```
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 ini adalah implementasi dari loop pelatihan (training loop) untuk melatih model regresi linear. Inisialisasi Seed: torch.manual_seed(42) Baris ini mengatur seed untuk generator angka acak PyTorch, sehingga hasil pelatihan dapat direproduksi.

Jumlah Epochs: epochs = 300 Menentukan jumlah iterasi pelatihan yang akan dilakukan.

Mengirim Data ke Perangkat Tertentu: X_train = X_train.to(device) X_test = X_test.to(device) y_train = y_train.to(device) y_test = y_test.to(device) Memastikan data dan target dikirim ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

Loop Pelatihan: for epoch in range(epochs): Melakukan iterasi sebanyak jumlah epochs yang telah ditentukan.

Pelatihan (Training): model_1.train() y_pred = model_1(X_train) loss = loss_fn(y_pred, y_train) optimizer.zero_grad() loss.backward() optimizer.step() model_1.train(): Menetapkan model ke mode pelatihan. y_pred: Melakukan prediksi dengan model pada data pelatihan. loss: Menghitung nilai loss antara prediksi dan target pelatihan. optimizer.zero_grad(): Mengatur gradien parameter model menjadi nol. loss.backward(): Menghitung gradien loss terhadap parameter model. optimizer.step(): Melakukan langkah optimisasi untuk memperbarui parameter model. Evaluasi pada Data Pengujian (Testing): if epoch % 20 == 0: model_1.eval() with torch.inference_mode(): y_preds = model_1(X_test) test_loss = loss_fn(y_preds, y_test) print(f"Epoch: {epoch} | Train loss: {loss:.3f} | Test loss: {test_loss:.3f}") Setiap 20 epoch, model dipindahkan ke mode evaluasi, dan dilakukan evaluasi pada data pengujian untuk mengukur kinerja model di luar data pelatihan. Hasil evaluasi termasuk informasi tentang loss pada data pelatihan dan pengujian, yang dicetak selama proses pelatihan. Dengan mencetak informasi ini, Anda dapat memantau bagaimana performa model berubah selama pelatihan.

```
# 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 ini digunakan untuk membuat prediksi dengan model yang telah dilatih pada data pengujian (X_test). model_1.eval() Baris pertama menggunakan metode .eval() untuk menetapkan model ke

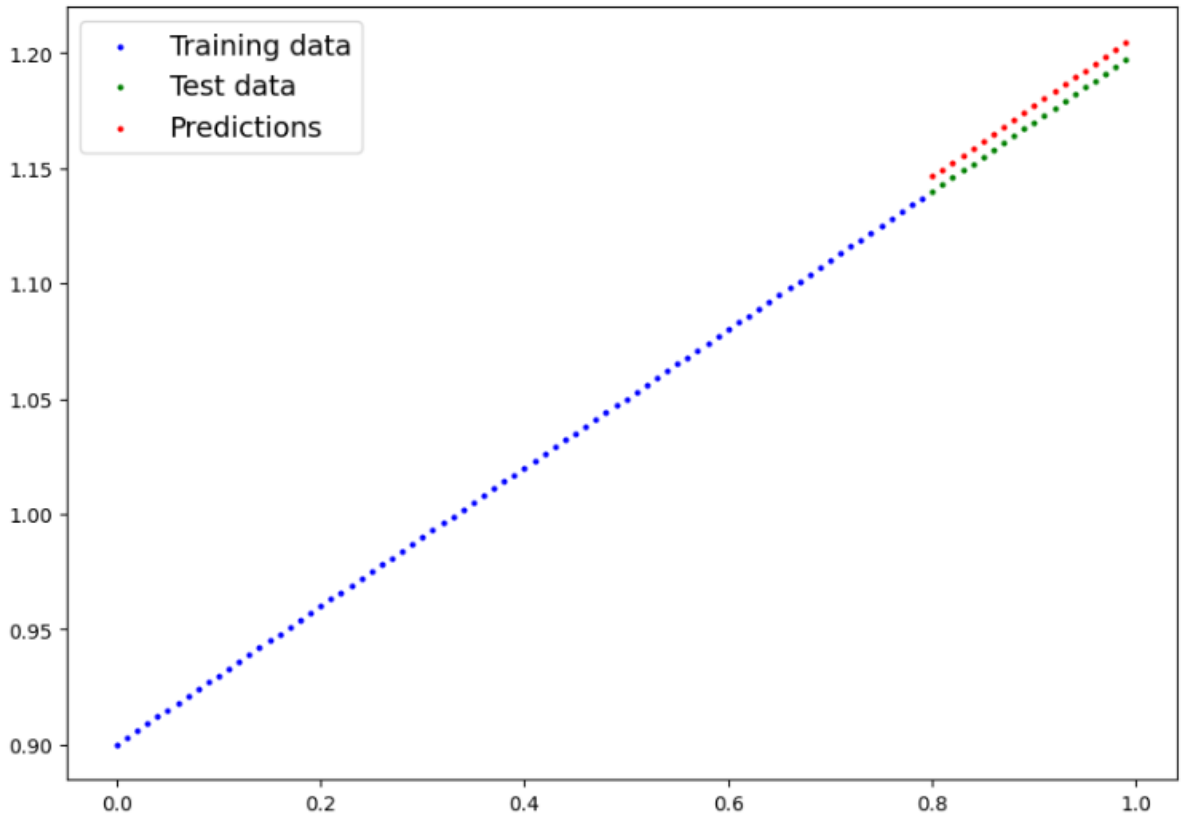
mode evaluasi. Mode evaluasi ini biasanya digunakan saat melakukan inferensi atau membuat prediksi. Ini memastikan bahwa layer-layer dalam model yang memiliki perilaku berbeda antara mode pelatihan dan evaluasi (seperti dropout atau batch normalization) akan berperilaku dengan benar. with torch.inference_mode(): y_preds = model_1(X_test) Baris ini menggunakan torch.inference_mode() untuk masuk ke mode inferensi saat melakukan prediksi. Meskipun sebelumnya kita telah menetapkan model ke mode evaluasi, penggunaan torch.inference_mode() memberikan jaminan tambahan bahwa operasi-inferensi khusus dapat dijalankan. y_preds Baris terakhir mencetak hasil prediksi yang dihasilkan oleh model pada 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]])
```

Outputnya berupa tensor yang berisi prediksi yang dihasilkan oleh model. Jika kita mencetak y_preds, kita akan melihat nilai-nilai numerik yang merupakan prediksi model pada setiap sampel dari data pengujian.

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



Kode ini digunakan untuk membuat plot dari hasil prediksi yang telah dihasilkan oleh model pada data pengujian. plot_predictions(predictions=y_preds.cpu()):

Memanggil fungsi plot_predictions() dengan menyertakan parameter predictions yang berisi tensor prediksi (y_preds) setelah diubah ke CPU menggunakan .cpu(). Hal ini dilakukan karena fungsi plot_predictions() kemungkinan besar dirancang untuk menangani tensor di CPU, dan ini akan memastikan bahwa hasilnya dapat diplot dengan benar. y_preds.cpu():

Menggunakan .cpu() untuk memindahkan tensor prediksi dari GPU ke CPU. Sebelumnya, prediksi model (y_preds) dihasilkan pada perangkat CUDA (GPU) (device='cuda:0'). Pemindahan ke CPU diperlukan agar hasilnya dapat digunakan dengan fungsi plotting yang mungkin tidak mendukung operasi pada GPU. plot_predictions():

Merupakan fungsi yang mungkin telah didefinisikan sebelumnya untuk memvisualisasikan hasil prediksi model. Fungsi ini menerima argumen seperti data pelatihan, data pengujian, dan hasil prediksi, dan kemungkinan besar menggunakan Matplotlib atau library plotting lainnya untuk membuat plot.

Hasil output dari fungsi plotting ini adalah sebuah plot yang menunjukkan bagaimana prediksi model membandingkan dengan data sebenarnya pada data pengujian. Plot tersebut mungkin mencakup titik-

titik biru untuk data pelatihan, titik-titik hijau untuk data pengujian, dan titik-titik merah untuk hasil prediksi model.

Plot yang dihasilkan bisa berupa scatter plot di mana sumbu x adalah nilai input (X_test), sumbu y adalah nilai output yang sebenarnya (y_test), dan titik-titik merah menunjukkan hasil prediksi model. Plot ini membantu memvisualisasikan seberapa baik model dapat memodelkan pola dari data dan sejauh mana prediksi model cocok dengan data yang sebenarnya.

```
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 ini digunakan untuk menyimpan state dictionary dari model PyTorch ke dalam file. from pathlib import Path Baris pertama mengimpor modul Path dari pustaka pathlib. Modul ini digunakan untuk bekerja dengan jalur (path) file dan direktori di sistem file. MODEL_PATH = Path("models") MODEL_PATH.mkdir(parents=True, exist_ok=True) Baris ini membuat direktori (folder) dengan nama "models" menggunakan objek Path. Jika direktori tersebut sudah ada, kode ini akan tetap berjalan tanpa menghasilkan kesalahan (exist_ok=True). MODEL_NAME = "01_pytorch_model" MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME Baris ini menentukan nama model (MODEL_NAME) dan membuat jalur lengkap ke tempat penyimpanan model (MODEL_SAVE_PATH). Jalur lengkap dibentuk dengan menggabungkan MODEL_PATH dan MODEL_NAME menggunakan operator / di pustaka pathlib. print(f"Saving model to {MODEL_SAVE_PATH}") torch.save(obj=model_1.state_dict(), f=MODEL_SAVE_PATH) Baris ini menyimpan state dictionary dari model (model_1) ke dalam file pada jalur yang telah ditentukan (MODEL_SAVE_PATH). Fungsi torch.save() digunakan untuk menyimpan objek PyTorch ke dalam file. Dalam hal ini, model_1.state_dict() berisi parameter-parameter dari model, dan MODEL_SAVE_PATH adalah jalur tempat menyimpannya.

Output dari print akan mencetak jalur lengkap tempat model disimpan.

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

```
LinearRegressionModel()
```

Kode ini digunakan untuk membuat instansi baru dari model regresi linear, dan kemudian memuat state dictionary yang telah disimpan sebelumnya ke dalam model tersebut. loaded_model = LinearRegressionModel():

Membuat instansi baru dari model regresi linear dengan menggunakan kelas LinearRegressionModel(). Model ini akan memiliki parameter-parameter yang diinisialisasi secara acak. loaded_model.load_state_dict(torch.load(f=MODEL_SAVE_PATH)):

Memuat state dictionary yang telah disimpan sebelumnya dari file ke dalam model baru (loaded_model). Fungsi torch.load() digunakan untuk membaca file yang berisi state dictionary, dan load_state_dict() digunakan untuk memuatnya ke dalam model. loaded_model.to(device):

Memindahkan model (loaded_model) ke perangkat yang telah ditentukan sebelumnya (device). Ini penting jika model dan data akan dioperasikan di GPU, karena state dictionary yang di-load mungkin berasal dari pelatihan di GPU.

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

Kode ini digunakan untuk membuat prediksi menggunakan model yang telah dilatih sebelumnya (yang baru saja dimuat dari state dictionary), dan kemudian membandingkan prediksi tersebut dengan prediksi yang telah dihasilkan sebelumnya menggunakan model awal. y_preds_new = loaded_model(X_test):

Menggunakan model yang telah dilatih sebelumnya (loaded_model) untuk membuat prediksi pada data pengujian (X_test). Hasilnya disimpan dalam variabel y_preds_new. y_preds == y_preds_new:

Membandingkan prediksi yang dihasilkan oleh model awal (y_preds) dengan prediksi yang dihasilkan oleh model yang baru dimuat (y_preds_new). Operasi perbandingan ini akan menghasilkan tensor Boolean yang menunjukkan apakah setiap elemen dari y_preds sama dengan elemen yang sesuai dari y_preds_new.

Output dari operasi perbandingan ini bisa berupa tensor Boolean yang menunjukkan kesamaan atau ketidakkesamaan antara kedua set prediksi. perbandingan menggunakan == pada tensor PyTorch akan menghasilkan tensor Boolean dengan elemen-elemen yang menunjukkan apakah elemen-elemen yang sesuai sama atau tidak. Jika hasilnya adalah tensor True pada semua elemen, itu berarti prediksi dari model yang awal dan model yang dimuat identik. Jika terdapat False, ada perbedaan antara kedua set prediksi.

```
loaded_model.state_dict()
```

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

Kode loaded_model.state_dict() digunakan untuk mengakses state dictionary dari model yang telah dimuat. loaded_model:

Merupakan objek model (dalam hal ini, model regresi linear) yang telah dimuat sebelumnya dari state dictionary. .state_dict():

Metode ini digunakan untuk mengambil state dictionary dari model. State dictionary adalah sebuah dictionary (kumpulan pasangan kunci dan nilai) yang berisi semua parameter dan gradien dari model. State dictionary ini berperan penting dalam menyimpan dan memuat model, karena berisi informasi lengkap tentang parameter-parameter model yang telah dilatih.

Output ini menunjukkan bahwa state dictionary berisi dua entri: satu untuk parameter weight dan satu untuk parameter bias. Tensor-tensor ini adalah nilai-nilai aktual yang dipelajari oleh model selama pelatihan.

Kita dapat menggunakan loaded_model.state_dict() untuk menyimpan state dictionary ke dalam file atau untuk mengeksplorasi nilai-nilai parameter yang ada pada model yang telah dimuat.