

TECHNICAL REPORT MACHINE LEARNING

Pytorch for Deep Learning



Disusun oleh:

Ario Syawal Muhammad / 1103201243

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

2024

A. PYTORCH

PyTorch adalah sebuah framework open-source untuk pengembangan model deep learning. Framework ini dikembangkan oleh Facebook's AI Research lab (FAIR) dan dirilis pada tahun 2016. PyTorch dirancang untuk mendukung pengembangan model deep learning dengan fokus pada fleksibilitas dan ekspresivitas.

Beberapa kegunaan utama PyTorch meliputi:

1. Pengembangan Model Deep Learning
2. Pembelajaran Mesin
3. Penelitian dan Pengembangan AI
4. Komputasi Ilmiah
5. Proyek Ilmiah
6. Proyek-proyek Pemrosesan Citra dan Visi Komputer
7. Pengembangan Model di Lingkungan Forensik

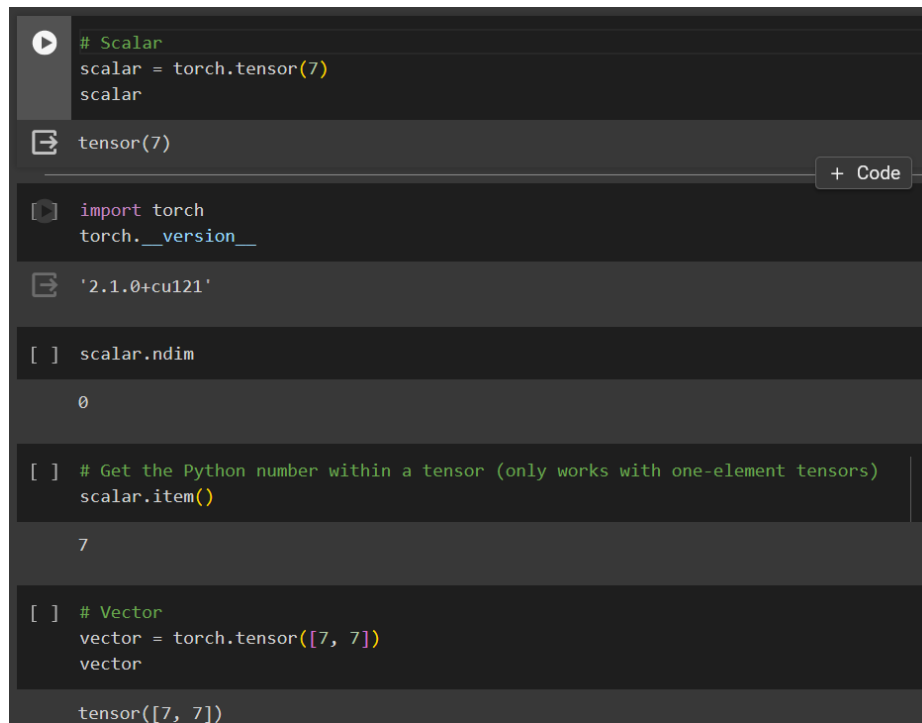
B. Zero to Mastery Learn PyTorch for Deep Learning

Kursus ini dibagi menjadi beberapa bagian (notebook). Setiap notebook mencakup ide dan konsep penting dalam PyTorch. Notebook berikutnya membangun pengetahuan dari yang sebelumnya (penomoran dimulai dari 00, 01, 02, dan seterusnya sesuai dengan kebutuhan). Notebook ini membahas blok dasar dari machine learning dan deep learning, yaitu tensor.

Chapter yang akan dibahas adalah :

1. Chapter 00 PyTorch Fundamentals
2. Chapter 01 PyTorch Workflow Fundamentals
3. Chapter 02 PyTorch Neural Network Classification
4. Chapter 03 PyTorch Computer Vision

❖ CHAPTER 00 PyTorch Fundamentals



```
# Scalar
scalar = torch.tensor(7)
scalar

tensor(7)

import torch
torch.__version__

'2.1.0+cu121'

[ ] scalar.ndim

0

[ ] # Get the Python number within a tensor (only works with one-element tensors)
    scalar.item()

7

[ ] # Vector
    vector = torch.tensor([7, 7])
    vector

tensor([7, 7])
```

Pada gambar diatas kita bisa melihat nilai tensor scalar yaitu 7, sekaligus mengimpor library PyTorch dan menampilkan versinya. Dengan menggunakan `torch.__version__`, kita dapat memeriksa versi PyTorch yang sedang digunakan. **ndim** adalah atribut yang digunakan untuk mengetahui jumlah dimensi dari tensor. Namun, skalar hanya memiliki satu elemen, sehingga dimensinya adalah 0. Sehingga `scalar.ndim` akan menghasilkan nilai 0. **Metode `item()`** digunakan untuk mengambil nilai skalar dari tensor. Dalam hal ini, itu akan mengembalikan nilai 7. Lalu kita juga membuat tensor vektor dengan dua elemen, yaitu `[7, 7]`. Vektor adalah tensor dengan satu dimensi.

```
# Check the number of dimensions of vector
vector.ndim

1

[ ] # Check shape of vector
vector.shape

torch.Size([2])

[ ]

# Matrix
MATRIX = torch.tensor([[7, 8],
                        [9, 10]])
MATRIX

tensor([[ 7,  8],
        [ 9, 10]])

[ ] # Check number of dimensions
MATRIX.ndim

2

[ ] MATRIX.shape

torch.Size([2, 2])
```

vector.ndim menghasilkan nilai 1 karena vektor adalah tensor satu dimensi, dan **vector.shape** akan memberikan output **torch.Size([2])**, menunjukkan bahwa vektor ini memiliki panjang 2. Di sini, kita membuat sebuah tensor matriks dengan dua dimensi menggunakan PyTorch. Matriks ini memiliki dua baris dan dua kolom. **MATRIX.ndim** memberikan nilai 2 karena matriks adalah tensor dua dimensi.

```
# TENSOR
TENSOR = torch.tensor([[[1, 2, 3],
                        [3, 6, 9],
                        [2, 4, 5]]])

TENSOR

tensor([[[1, 2, 3],
        [3, 6, 9],
        [2, 4, 5]]])

[ ] # Check number of dimensions for TENSOR
TENSOR.ndim

3

[ ] # Check shape of TENSOR
TENSOR.shape

torch.Size([1, 3, 3])

[ ] # Create a random tensor of size (3, 4)
random_tensor = torch.rand(size=(3, 4))
random_tensor, random_tensor.dtype

(tensor([[0.1415, 0.0296, 0.8362, 0.6155],
        [0.2637, 0.2687, 0.3492, 0.2293],
        [0.0434, 0.1519, 0.1366, 0.8618]]),
 torch.float32)

[ ] # Create a random tensor of size (224, 224, 3)
random_image_size_tensor = torch.rand(size=(224, 224, 3))
random_image_size_tensor.shape, random_image_size_tensor.ndim
```

Di sini, kita membuat tensor 3D dengan menggunakan PyTorch. Tensor ini memiliki tiga dimensi dan berisi elemen-elemen yang diberikan. `TENSOR.ndim` memberikan nilai 3, menunjukkan bahwa ini adalah tensor tiga dimensi. `TENSOR.shape` menghasilkan output `torch.Size([1, 3, 3])`, menunjukkan bahwa tensor ini memiliki panjang 1 di dimensi pertama, panjang 3 di dimensi kedua, dan panjang 3 di dimensi ketiga.

Menggunakan fungsi `torch.rand()` kita bisa membuat tensor acak dengan ukuran (3, 4). Ini akan menghasilkan tensor dengan nilai acak antara 0 dan 1. `random_image_size_tensor` mencoba membuat tensor acak dengan ukuran (224, 224, 3). Namun, perlu dicatat bahwa tensor ini tidak sesuai untuk merepresentasikan gambar, karena dalam dunia deep learning, dimensi terakhir umumnya harus mewakili saluran warna (seperti RGB) dan bukan panjang dimensi.

```
# Create a tensor of all zeros
zeros = torch.zeros(size=(3, 4))
zeros, zeros.dtype

(tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]),
 torch.float32)
```

Kode di atas membuat sebuah tensor dengan seluruh elemennya bernilai nol. Sehingga, zeros adalah tensor dengan ukuran (3, 4) yang diisi dengan nilai nol dan memiliki tipe data float32. Penciptaan tensor dengan nilai nol umumnya berguna sebagai inisialisasi awal untuk parameter model atau saat membutuhkan tensor dengan nilai tetap.

```
# Create a tensor of all ones
ones = torch.ones(size=(3, 4))
ones, ones.dtype

(tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]),
 torch.float32)
```

Dengan menggunakan fungsi torch.ones(), kita membuat tensor dengan ukuran (3, 4) yang seluruh elemennya diinisialisasi dengan nilai satu. Sehingga, ones adalah tensor dengan ukuran (3, 4) yang diisi dengan nilai satu dan memiliki tipe data float32. Penciptaan tensor dengan nilai satu umumnya berguna sebagai inisialisasi awal untuk parameter model atau saat membutuhkan tensor dengan nilai tetap.

```
[ ] # Use torch.arange(), torch.range() is deprecated
zero_to_ten_deprecated = torch.range(0, 10) # Note: this

# Create a range of values 0 to 10
zero_to_ten = torch.arange(start=0, end=10, step=1)
zero_to_ten

<ipython-input-19-a09072c806d9>:2: UserWarning: torch.range
zero_to_ten_deprecated = torch.range(0, 10) # Note: th
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Program di atas menggunakan PyTorch untuk membuat tensor yang berisi rentang nilai dari 0 hingga 9. Perlu diperhatikan bahwa penggunaan torch.range() adalah metode lama dan sudah dianggap usang (deprecated). Penggunaannya masih dapat berfungsi saat ini, tetapi dapat menghasilkan pesan peringatan dan mungkin tidak didukung di masa mendatang. Oleh karena itu, sebaiknya diganti dengan fungsi yang lebih baru seperti torch.arange() untuk membuat tensor yang berisi rentang nilai dari 0 hingga 9 dengan langkah 1.

```
# Can also create a tensor of zeros similar to a
ten_zeros = torch.zeros_like(input=zero_to_ten)
ten_zeros

tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Dengan menggunakan torch.zeros_like(), kita membuat tensor dengan ukuran dan tipe data yang sama dengan tensor zero_to_ten, tetapi diisi dengan nilai nol. Jadi, kita berhasil membuat tensor yang berisi nilai nol, tetapi memiliki bentuk yang sama dengan tensor zero_to_ten. Ini bisa berguna saat kita ingin membuat tensor dengan inisialisasi awal tetapi dengan bentuk yang sudah ditentukan.

```
# Default datatype for tensors is float32
float_32_tensor = torch.tensor([3.0, 6.0, 9.0],
                                dtype=None, # defaults to None, which
                                device=None, # defaults to None, which
                                requires_grad=False) # if True, operat

float_32_tensor.shape, float_32_tensor.dtype, float_32_tensor.device
(torch.Size([3]), torch.float32, device(type='cpu'))
```

kita membuat tensor dengan nilai [3.0, 6.0, 9.0]. Pada baris kode ini, parameter dtype diatur ke None, yang secara default akan menggunakan tipe data float32. Oleh karena itu, tensor ini memiliki tipe data float32. Jadi, tensor float_32_tensor adalah tensor dengan nilai [3.0, 6.0, 9.0], tipe data float32, dan disimpan pada perangkat default.

```
float_16_tensor = torch.tensor([3.0, 6.0, 9.0],
                                dtype=torch.float16)

float_16_tensor.dtype
torch.float16
```

Program di atas menciptakan sebuah tensor PyTorch yang berisi nilai float (bilangan desimal) dengan tipe data float16. kita membuat tensor dengan nilai [3.0, 6.0, 9.0] dan secara eksplisit menetapkan tipe datanya ke torch.float16 atau torch.half. Oleh karena itu, tensor ini memiliki tipe data float16. float_16_tensor.dtype memberikan informasi tentang tipe data tensor, yang dalam hal ini adalah torch.float16.

```
# Create a tensor
some_tensor = torch.rand(3, 4)

# Find out details about it
print(some_tensor)
print(f"Shape of tensor: {some_tensor.shape}")
print(f"Datatype of tensor: {some_tensor.dtype}")
print(f"Device tensor is stored on: {some_tensor.device}")

tensor([[0.3756, 0.3758, 0.5469, 0.4841],
        [0.1347, 0.6794, 0.4063, 0.3640],
        [0.9171, 0.5267, 0.8637, 0.0976]])
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

Program di atas menciptakan tensor PyTorch yang berisi nilai acak (dari distribusi seragam) dengan ukuran 3x4 dan kemudian mencetak beberapa detail tentang tensor tersebut. Dengan menggunakan torch.rand(), kita membuat tensor dengan ukuran 3x4 yang diisi dengan nilai acak antara 0 dan 1 dari distribusi seragam. Jadi, output dari program ini akan memberikan informasi tentang bentuk, tipe data, dan perangkat tempat tensor disimpan.

```

# Create a tensor of values and add a number to it
tensor = torch.tensor([1, 2, 3])
tensor + 10

tensor([11, 12, 13])

# Multiply it by 10
tensor * 10

tensor([10, 20, 30])

# Tensors don't change unless reassigned
tensor

tensor([1, 2, 3])

# Subtract and reassign
tensor = tensor - 10
tensor

tensor([-9, -8, -7])

# Add and reassign
tensor = tensor + 10
tensor

tensor([1, 2, 3])

# Can also use torch functions
torch.multiply(tensor, 10)

tensor([10, 20, 30])

```

Program di atas menggunakan PyTorch untuk membuat tensor, melakukan operasi matematika pada tensor tersebut, dan menunjukkan bagaimana merubah nilai tensor dengan menetapkan ulang. Kita membuat tensor dengan nilai [1, 2, 3] dan menambahkan 10 ke setiap elemennya. Ini adalah contoh operasi matematika elemen demi elemen pada tensor. Meskipun kita melakukan operasi penambahan dan perkalian, nilai tensor tidak berubah kecuali kita menetapkannya kembali. Sebagai alternatif, kita dapat menggunakan fungsi PyTorch seperti `torch.multiply()` untuk mengalikan setiap elemen tensor dengan nilai tertentu.


```
0s ✓ # Original tensor is still unchanged
tensor

tensor([1, 2, 3])

[74] ✓ # Element-wise multiplication (each element
print(tensor, "* ", tensor)
print("Equals:", tensor * tensor)

tensor([1, 2, 3]) * tensor([1, 2, 3])
Equals: tensor([1, 4, 9])

[75] ✓ import torch
tensor = torch.tensor([1, 2, 3])
tensor.shape

torch.Size([3])

[76] ✓ # Element-wise matrix multiplication
tensor * tensor

tensor([1, 4, 9])

[77] ✓ # Matrix multiplication
torch.matmul(tensor, tensor)

tensor(14)
```

Operasi `tensor * tensor` menghasilkan perkalian elemen demi elemen. Setiap elemen pada posisi yang sama di kedua tensor dikalikan satu sama lain. tensor saat ini adalah tensor satu dimensi dengan panjang 3. Fungsi `torch.matmul()` digunakan untuk melakukan perkalian matriks dari tensor dengan dirinya sendiri. Perlu diperhatikan bahwa untuk perkalian matriks yang benar, dimensi tensor harus sesuai (misalnya, tensor 1D dengan tensor 1D).

```
%%time
# Matrix multiplication by hand
# (avoid doing operations with for loops at all cost, t
value = 0
for i in range(len(tensor)):
    value += tensor[i] * tensor[i]
value

CPU times: user 289 µs, sys: 0 ns, total: 289 µs
Wall time: 298 µs
tensor(14)
```

Dengan menggunakan magic command `%%time`, kita dapat mengukur waktu yang diperlukan untuk menjalankan sel tersebut.

```
[113] # Shapes need to be in the right way
tensor_A = torch.tensor([[1, 2],
                        [3, 4],
                        [5, 6]], dtype=torch.float32)

tensor_B = torch.tensor([[7, 10],
                        [8, 11],
                        [9, 12]], dtype=torch.float32)

torch.matmul(tensor_A, tensor_B) # (this will error)
```

Program di atas adalah contoh penggunaan PyTorch untuk membuat dua tensor (tensor_A dan tensor_B), yang masing-masing berisi data matriks 3x2 dengan tipe data float32. Selanjutnya, program mencoba melakukan perkalian matriks antara tensor_A dan tensor_B.

```
# View tensor_A and tensor_B
print(tensor_A)
print(tensor_B)

tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])
tensor([[ 7., 10.],
        [ 8., 11.],
        [ 9., 12.]])

[115] # View tensor_A and tensor_B.T
print(tensor_A)
print(tensor_B.T)

tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])
tensor([[ 7.,  8.,  9.],
        [10., 11., 12.]])
```

Pertama-tama, matriks tensor_A dan tensor_B ditampilkan. Kemudian, matriks tensor_B.T ditampilkan, yang merupakan transposisi dari tensor_B. Transposisi dilakukan dengan menukar baris dan kolom, sehingga kolom pada tensor_B menjadi baris pada tensor_B.T.

```
# Since the linear layer starts with a random weights matrix,
torch.manual_seed(42)
# This uses matrix multiplication
linear = torch.nn.Linear(in_features=2, # in_features = matche
                        out_features=6) # out_features = desc
x = tensor_A
output = linear(x)
print(f"Input shape: {x.shape}\n")
print(f"Output:\n{output}\n\nOutput shape: {output.shape}")
```

Input shape: torch.Size([3, 2])

Output:

```
tensor([[2.2368, 1.2292, 0.4714, 0.3864, 0.1309, 0.9838],
        [4.4919, 2.1970, 0.4469, 0.5285, 0.3401, 2.4777],
        [6.7469, 3.1648, 0.4224, 0.6705, 0.5493, 3.9716]],
       grad_fn=<AddmmBackward0>)
```

Output shape: torch.Size([3, 6])

Program di atas menggunakan PyTorch untuk membuat layer linier dengan parameter tertentu, dan kemudian mengaplikasikan layer linier tersebut pada tensor `tensor_A`. Program juga mencoba membuat inisialisasi pembobotan linier menjadi reproducible (dapat direproduksi) dengan mengatur seed (benih) generator angka acak menggunakan `torch.manual_seed(42)`. Sebagai catatan, pengaturan seed pada `torch.manual_seed(42)` memberikan asumsi bahwa library lainnya yang digunakan juga diatur untuk seed yang sama agar hasilnya dapat sepenuhnya direproduksi.

```
# Create a tensor
x = torch.arange(0, 100, 10)
x
```

tensor([0, 10, 20, 30, 40, 50, 60, 70, 80, 90])

```
print(f"Minimum: {x.min()}")
print(f"Maximum: {x.max()}")
# print(f"Mean: {x.mean()}") # this will error
print(f"Mean: {x.type(torch.float32).mean()}") # v
print(f"Sum: {x.sum()}")
```

Minimum: 0
Maximum: 90
Mean: 45.0
Sum: 450

Program di atas menggunakan PyTorch untuk membuat tensor `x` yang berisi nilai dari 0 hingga 90 dengan selang 10. Program kemudian mencetak nilai minimum, maksimum, rata-rata, dan jumlah dari tensor tersebut. Dengan menggunakan `torch.arange()`, kita membuat tensor `x` yang berisi nilai dari 0 hingga 90 (tidak termasuk 100) dengan selang 10. Program mencetak nilai minimum dan maksimum dari tensor `x` menggunakan metode `min()` dan `max()`. Pemanggilan `x.mean()` akan menghasilkan error karena defaultnya, `torch.arange()` membuat tensor dengan tipe data long integer (`int64`)

```
[88] torch.max(x), torch.min(x), torch.mean(x.type(torch.float32)), torch.sum(x)

(tensor(90), tensor(0), tensor(45.), tensor(450))

# Create a tensor
tensor = torch.arange(10, 100, 10)
print(f"Tensor: {tensor}")

# Returns index of max and min values
print(f"Index where max value occurs: {tensor.argmax()}")
print(f"Index where min value occurs: {tensor.argmin()}")

Tensor: tensor([10, 20, 30, 40, 50, 60, 70, 80, 90])
Index where max value occurs: 8
Index where min value occurs: 0
```

fungsi `torch.max()`, `torch.min()`, `torch.mean()`, dan `torch.sum()` pada tensor `x` untuk menghitung nilai maksimum, nilai minimum, rata-rata, dan jumlah dari tensor tersebut. membuat tensor baru dengan menggunakan `torch.arange()` dan mencetak nilai tensor tersebut. metode `.argmax()` dan `.argmin()` pada tensor tensor untuk mendapatkan indeks di mana nilai maksimum dan minimum terjadi.

```
# Create a tensor and check its datatype
tensor = torch.arange(10., 100., 10.)
tensor.dtype

torch.float32
```

Dalam blok kode ini, membuat tensor baru dengan menggunakan `torch.arange()` dengan nilai awal 10.0, nilai akhir 100.0, dan selang 10.0. Setelah itu, Anda memeriksa tipe data (`dtype`) dari tensor tersebut. Dengan menggunakan `torch.arange()`, membuat tensor tensor yang berisi nilai dari 10 hingga 90 (tidak termasuk 100) dengan selang 10.0. Penggunaan desimal (.) pada angka memastikan bahwa tipe data tensor adalah float.

```
# Create a float16 tensor
tensor_float16 = tensor.type(torch.float16)
tensor_float16

tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.])
```

Dalam blok kode ini, mengonversi tensor tensor yang telah dibuat sebelumnya menjadi tensor dengan tipe data float16 menggunakan metode `type()`.

```
# Create a int8 tensor
tensor_int8 = tensor.type(torch.int8)
tensor_int8

tensor([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

mencoba mengonversi tensor tensor yang telah dibuat sebelumnya menjadi tensor dengan tipe data int8 menggunakan metode `type()`. Namun, perlu diingat bahwa pada umumnya, PyTorch tidak mendukung tipe data int8 secara langsung. Sehingga, hasilnya mungkin bukan int8

melainkan int16 atau int32, tergantung pada konfigurasi PyTorch dan perangkat keras yang digunakan.

```
[ ] # Create a tensor
import torch
x = torch.arange(1., 8.)
x, x.shape

(tensor([1., 2., 3., 4., 5., 6., 7.]), torch.Size([7]))

# Add an extra dimension
x_resaped = x.reshape(1, 7)
x_resaped, x_resaped.shape

(tensor([[1., 2., 3., 4., 5., 6., 7.]]), torch.Size([1, 7]))

# Change view (keeps same data as original but changes
# See more: https://stackoverflow.com/a/54507446/790072)
z = x.view(1, 7)
z, z.shape

(tensor([[1., 2., 3., 4., 5., 6., 7.]]), torch.Size([1, 7]))

# Changing z changes x
z[:, 0] = 5
z, x

(tensor([[5., 2., 3., 4., 5., 6., 7.]]), tensor([5., 2., 3., 4., 5., 6., 7.]))
```

membuat tensor x dengan nilai dari 1 hingga 7. Kemudian, program menambahkan dimensi ekstra ke tensor tersebut, mengubah tampilan tensor, dan menunjukkan bagaimana perubahan pada tensor z juga mempengaruhi tensor x. Dengan menggunakan `torch.arange()`, Anda membuat tensor x yang berisi nilai dari 1 hingga 7. menggunakan metode `view()`, Anda mengubah tampilan tensor x tanpa mengubah data yang sebenarnya. Hasilnya adalah tensor dengan bentuk (1, 7).

```
] # Stack tensors on top of each other
x_stacked = torch.stack([x, x, x, x], dim=0) # t
x_stacked

(tensor([[[5., 2., 3., 4., 5., 6., 7.],
         [5., 2., 3., 4., 5., 6., 7.],
         [5., 2., 3., 4., 5., 6., 7.],
         [5., 2., 3., 4., 5., 6., 7.]]]),
```

menggunakan fungsi `torch.stack()` untuk menggabungkan tensor x sebanyak empat kali secara vertikal. Dimensi stack ditentukan dengan parameter `dim`. Fungsi `torch.stack()` digunakan untuk menggabungkan tensor x sebanyak empat kali.

```
# Create a tensor
import torch
x = torch.arange(1, 10).reshape(1, 3, 3)
x, x.shape

(tensor([[[[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]]]),
 torch.Size([1, 3, 3]))
```

membuat tensor x dengan menggunakan fungsi `torch.arange()` untuk menghasilkan nilai dari 1 hingga 9, dan kemudian mengubah bentuknya menjadi (1, 3, 3).

```
import torch
import random

# Set the random seed
RANDOM_SEED=42 # try changing this to different values and
torch.manual_seed(seed=RANDOM_SEED)
random_tensor_C = torch.rand(3, 4)

# Have to reset the seed every time a new rand() is called
# Without this, tensor_D would be different to tensor_C
torch.random.manual_seed(seed=RANDOM_SEED) # try commenting
random_tensor_D = torch.rand(3, 4)

print(f"Tensor C:\n{random_tensor_C}\n")
print(f"Tensor D:\n{random_tensor_D}\n")
print(f"Does Tensor C equal Tensor D? (anywhere)")
random_tensor_C == random_tensor_D
```

dua tensor acak (`random_tensor_C` dan `random_tensor_D`) dengan dimensi 3x4. Anda juga mengatur biji acak (`torch.manual_seed()`) untuk memastikan bahwa hasilnya dapat direproduksi. menggunakan `torch.manual_seed()` dengan nilai seed 42. Ini memastikan bahwa angka acak yang dihasilkan oleh PyTorch akan sama setiap kali menjalankan program ini.

❖ CHAPTER 01 PyTorch Workflow Fundamentals

```
# Create *known* parameters
weight = 0.7
bias = 0.3

# Create data
start = 0
end = 1
step = 0.02
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias

X[:10], y[:10]
```

(tensor([[0.0000],

parameter-parameter yang diketahui (`weight` dan `bias`) serta menciptakan data dengan menggunakan persamaan linier. Data yang dihasilkan terdiri dari nilai-nilai X yang berurutan dari 0 hingga 1 dengan selang 0.02, dan nilai-nilai y yang dihitung dengan persamaan linier $\text{weight} * X + \text{bias}$. menggunakan `torch.arange()` untuk membuat tensor X yang berisi nilai-nilai yang berurutan dari 0 hingga 1 dengan selang 0.02. Kemudian, Anda menggunakan `unsqueeze(dim=1)` untuk menambahkan dimensi tambahan sehingga X menjadi tensor kolom. Hasilnya adalah tensor X yang berisi nilai-nilai dari 0 hingga 1 dengan selang 0.02 dan tensor y yang dihitung dengan menggunakan persamaan linier.

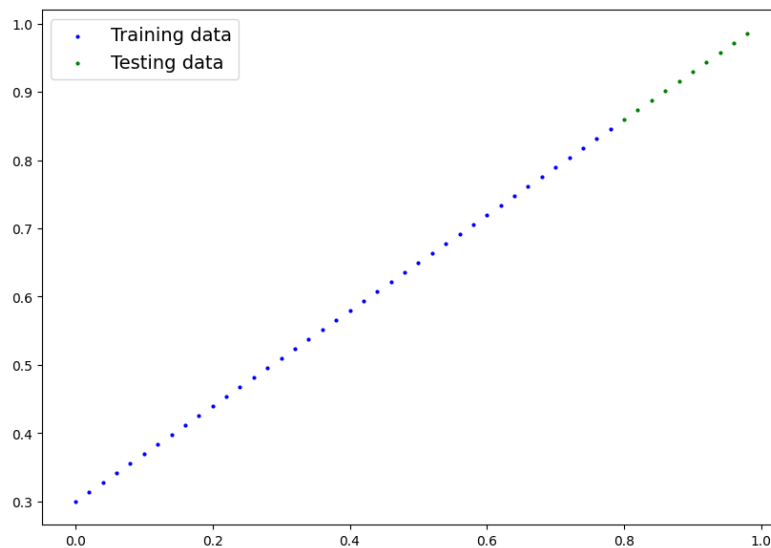
```

# Create train/test split
train_split = int(0.8 * len(X)) # 80% of data used for training
X_train, y_train = X[:train_split], y[:train_split]
X_test, y_test = X[train_split:], y[train_split:]

len(X_train), len(y_train), len(X_test), len(y_test)
(40, 40, 10, 10)

```

membuat pembagian data menjadi set pelatihan dan pengujian. Pembagian ini dilakukan dengan mengambil 80% data untuk set pelatihan dan 20% untuk set pengujian. menggunakan indeks train_split untuk membagi tensor X dan y menjadi dua set: set pelatihan dan set pengujian.



Fungsi plot_predictions yang didefinisikan di atas adalah fungsi untuk membuat plot dari data pelatihan, data pengujian, dan prediksi model (jika ada).

```

# Create a Linear Regression model class
class LinearRegressionModel(nn.Module): # <- almost every
    def __init__(self):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(1, # <- standard deviation
                                                dtype=torch.float,
                                                requires_grad=True) # <- requires gradient

        self.bias = nn.Parameter(torch.randn(1, # <- standard deviation
                                              dtype=torch.float,
                                              requires_grad=True) # <- requires gradient

    # Forward defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.weights * x + self.bias # <- this is the computation

```

mendefinisikan kelas model Regresi Linier menggunakan PyTorch. Ini adalah contoh sederhana dari kelas model yang menggunakan PyTorch's nn.Module. Dalam metode __init__, Anda menginisialisasi dua parameter model: weights dan bias. Kedua parameter ini dimulai dengan nilai acak (torch.randn()) dan diatur untuk dapat diubah nilainya (requires_grad=True), yang berarti PyTorch dapat memperbarui nilai-nilai ini selama proses pembelajaran. Metode

forward mendefinisikan komputasi yang dilakukan oleh model saat melakukan prediksi. Dalam hal ini, model mengembalikan hasil dari operasi regresi linier: $\text{self.weights} * x + \text{self.bias}$.

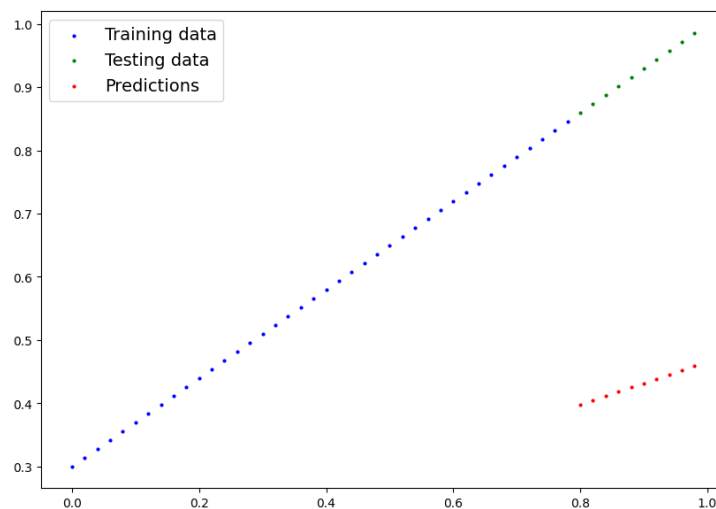
```
# List named parameters
model_0.state_dict()

OrderedDict([('weights', tensor([0.

[ ] # Make predictions with model
with torch.inference_mode():
    y_preds = model_0(X_test)

# Note: in older PyTorch code you
# with torch.no_grad():
#     y_preds = model_0(X_test)
```

menggunakan PyTorch untuk mengakses parameter-parameter yang dinamai dari model dan membuat prediksi dengan model tersebut. Dengan memanggil metode `state_dict()` pada model (`model_0`), Anda dapat mengakses dictionary yang berisi seluruh parameter dinamai dan nilai-nilai mereka.



Berikut hasil prediksi model pada data pengujian. terdapat 10 sampel dalam data pengujian (`X_test`), dan model membuat 10 prediksi untuk data pengujian.


```

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

    # Put model in training mode (this is the default state of a model)
    model_0.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model_0(X_train)
    # print(y_pred)

    # 2. Calculate the loss (how different are our models predictions to the ground truth)
    loss = loss_fn(y_pred, y_train)

    # 3. Zero grad of the optimizer
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Progress the optimizer
    optimizer.step()

    ### Testing

    # Put the model in evaluation mode
    model_0.eval()

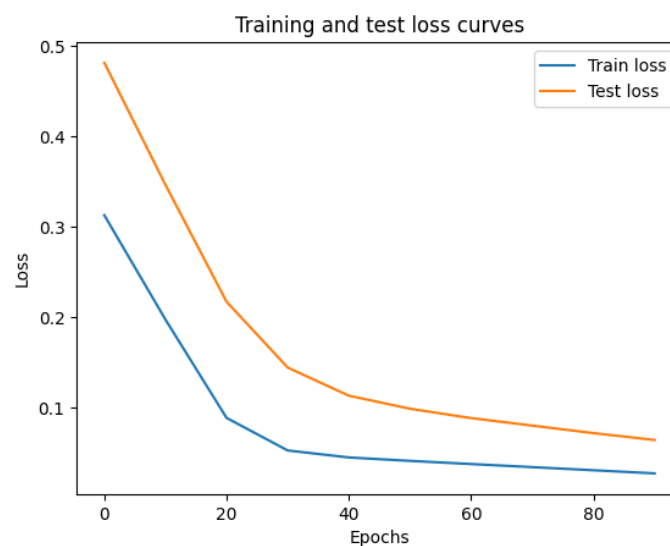
    with torch.inference_mode():
        # 1. Forward pass on test data
        test_pred = model_0(X_test)

        # 2. Caculate loss on test data
        test_loss = loss_fn(test_pred, y_test.type(torch.float)) # predictions come in torch.float

    # Print out what's happening
    if epoch % 10 == 0:
        epoch_count.append(epoch)
        train_loss_values.append(loss.detach().numpy())
        test_loss_values.append(test_loss.detach().numpy())
        print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")

```

implementasi sederhana dari proses pelatihan dan evaluasi model regresi linier menggunakan PyTorch. Model ditempatkan dalam mode pelatihan. Ini memastikan bahwa operasi seperti dropout (jika ada) berfungsi dengan benar selama pelatihan. Setiap 10 epoch, program mencetak informasi tentang loss pada data pelatihan dan pengujian. kita telah mendapatkan catatan tentang bagaimana loss model berubah selama proses pelatihan dan evaluasi. Catatan ini berguna untuk memahami sejauh mana model telah belajar dan apakah terdapat overfitting atau underfitting.



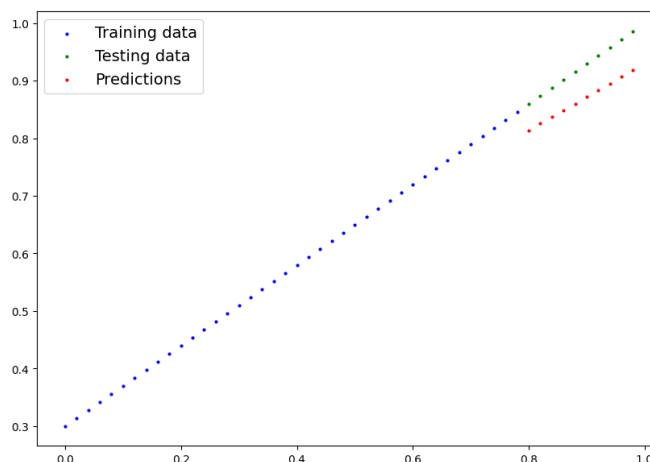
kurva loss pelatihan menurun secara signifikan seiring berjalannya waktu. Hal ini menunjukkan bahwa model belajar dan meningkatkan akurasi prediksinya. Kurva loss pengujian juga menurun pada awalnya, tetapi kemudian mulai naik. Hal ini menunjukkan bahwa model mulai overfitting data pelatihan.

```
# 1. Set the model in evaluation mode
model_0.eval()

# 2. Setup the inference mode context manager
with torch.inference_mode():
    # 3. Make sure the calculations are done with the model
    # in our case, we haven't setup device-agnostic code yet
    # on the CPU by default.
    # model_0.to(device)
    # X_test = X_test.to(device)
    y_preds = model_0(X_test)
y_preds
```

```
tensor([[0.8141],
        [0.8256],
        [0.8372],
        [0.8488],
        [0.8603],
        [0.8719],
        [0.8835],
        [0.8950],
        [0.9066],
        [0.9182]])
```

Dengan memanggil metode `eval()`, model ditempatkan dalam mode evaluasi. Ini memastikan bahwa operasi-operasi seperti dropout (jika ada) tidak berlaku selama inferensi. Dengan menggunakan `torch.inference_mode()`, Anda membuat konteks di mana PyTorch akan menangani operasi dengan cara yang lebih efisien selama inferensi. Ini dapat meningkatkan kinerja inferensi. memanggil model pada data pengujian (`X_test`), kita mendapatkan prediksi (`y_preds`) dari model pada data tersebut. Prediksi ini dapat digunakan untuk evaluasi lebih lanjut atau analisis hasil model.



dapat disimpulkan bahwa model tersebut mengalami overfitting setelah sekitar 30 epoch. Untuk meningkatkan kinerja model, perlu dilakukan langkah-langkah untuk mengurangi overfitting.

```
# 1. Put the loaded model into evaluation mode
loaded_model_0.eval()

# 2. Use the inference mode context manager to make
with torch.inference_mode():
    loaded_model_preds = loaded_model_0(X_test) # f

[ ] # Compare previous model predictions with loaded mo
y_preds == loaded_model_preds

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

Dengan memanggil metode eval(), model yang telah dimuat (loaded_model_0) ditempatkan dalam mode evaluasi. Ini memastikan bahwa operasi-operasi seperti dropout (jika ada) tidak berlaku selama inferensi. Kita mendapatkan prediksi (loaded_model_preds) dari model tersebut. Prediksi ini kemudian dapat digunakan untuk perbandingan atau evaluasi. Model yang dimuat disiapkan untuk membuat prediksi pada data pengujian dengan mode evaluasi dan konteks inferensi yang efisien.

```
# Create weight and bias
weight = 0.7
bias = 0.3

# Create range values
start = 0
end = 1
step = 0.02

# Create X and y (features and labels)
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias
X[:10], y[:10]

(tensor([[0.0000],
```

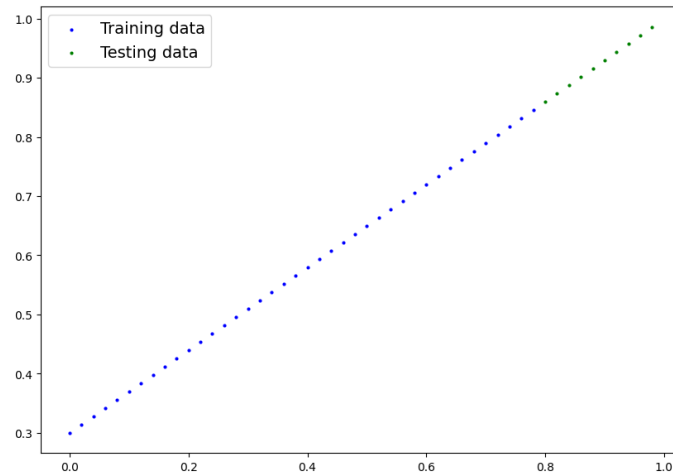
mendefinisikan nilai bobot (weight) dan bias (bias). Ini adalah parameter yang akan digunakan dalam fungsi regresi linier (yaitu, $y = \text{weight} * X + \text{bias}$). Nilai awal (start), nilai akhir (end), dan langkah (step) untuk membuat rentang nilai yang akan digunakan sebagai fitur (X) dalam contoh regresi. Dengan menggunakan unsqueeze(dim=1), Anda mengubah bentuk tensor agar sesuai dengan format yang diperlukan oleh model regresi linier (1 dimensi untuk setiap sampel). Selanjutnya, Anda menghitung label (y) dengan menggunakan rumus regresi linier menggunakan bobot dan bias yang telah ditentukan.

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

len(X_train), len(y_train), len(X_test), len(y_test)

(40, 40, 10, 10)
```

membagi data menjadi data pelatihan (X_{train} dan y_{train}) dan data pengujian (X_{test} dan y_{test}), data pelatihan akan mencakup 80% dari seluruh data. indeks slicing untuk mendapatkan 80% pertama dari data untuk digunakan sebagai data pelatihan, 20% terakhir dari data untuk digunakan sebagai data pengujian. Berikut adalah hasil dari pengujian.



```
# Create loss function
loss_fn = nn.L1Loss()

# Create optimizer
optimizer = torch.optim.SGD(params=model_1.parameters(),
                             lr=0.01)
```

menggunakan kelas `nn.L1Loss()` dari PyTorch untuk membuat fungsi kerugian. `L1Loss` mengukur kerugian absolut (absolute loss) antara setiap elemen dari prediksi dan target. Ini sering digunakan dalam regresi. menggunakan optimizer Stochastic Gradient Descent (SGD) dari PyTorch (`torch.optim.SGD`). Optimizer ini akan mengoptimalkan parameter-parameter model (`model_1.parameters()`) selama pelatihan dengan tingkat pembelajaran (`lr`) sebesar 0.01.

```

torch.manual_seed(42)

# Set the number of epochs
epochs = 1000

# Put data on the available device
# Without this, error will happen (not all model/data on 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
    model_1.train() # train mode is on by default after construction

    # 1. Forward pass
    y_pred = model_1(X_train)

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

    # 3. Zero grad optimizer
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Step the optimizer
    optimizer.step()

    ### Testing
    model_1.eval() # put the model in evaluation mode for testing (inference)
    # 1. Forward pass
    with torch.inference_mode():
        test_pred = model_1(X_test)

    # 2. Calculate the loss
    test_loss = loss_fn(test_pred, y_test)

    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Train loss: {loss} | Test loss: {test_loss}")

```

Blok kode di atas merupakan loop pelatihan (training loop) untuk model regresi linier (model_1). Dengan melakukan loop ini, melatih model regresi linier (model_1) untuk meminimalkan kerugian pada data pelatihan dan mengukur kinerja pada data pengujian selama sejumlah epochs yang ditentukan.

```

The model learned the following values for weights and bias:
OrderedDict([('linear_layer.weight', tensor([[0.6968]])),
            ('linear_layer.bias', tensor([0.3025]))])

And the original values for weights and bias are:
weights: 0.7, bias: 0.3

```

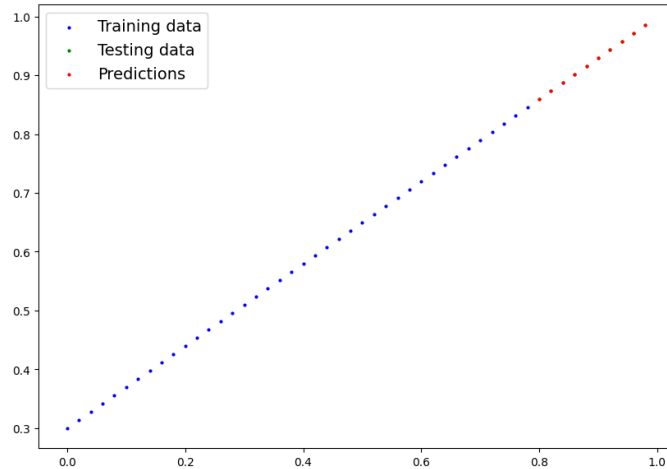
Hasil tersebut menunjukkan bahwa setelah pelatihan, model telah belajar untuk mendekati nilai-nilai yang diinginkan, yaitu mendekati bobot 0.7 dan bias 0.3. Proses ini menggambarkan bagaimana model linier dapat mengoptimalkan parameter (berat dan bias) untuk meminimalkan kerugian pada data pelatihan.

```

# Turn model into evaluation mode
model_1.eval()

# Make predictions on the test data
with torch.inference_mode():
    y_preds = model_1(X_test)
y_preds

```



Blok kode di atas mengubah model regresi linier (model_1) ke mode evaluasi dan kemudian membuat prediksi pada data pengujian (X_test).

```
# Evaluate loaded model
loaded_model_1.eval()
with torch.inference_mode():
    loaded_model_1_preds = loaded_model_1(X_test)
y_preds == loaded_model_1_preds

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

Dalam program sebelumnya, hasilnya adalah tensor yang seluruh elemennya True. Ini menunjukkan bahwa prediksi dari model (y_preds) dan prediksi dari model yang telah dimuat (loaded_model_1_preds) pada data pengujian adalah identik. Artinya, model yang telah dimuat memberikan hasil prediksi yang sama dengan model yang sedang digunakan, yang sesuai dengan harapan untuk model yang sama yang telah disimpan dan dimuat kembali.

❖ CHAPTER 02 PyTorch Neural Network Classification

```
from sklearn.datasets import make_circles

# Make 1000 samples
n_samples = 1000

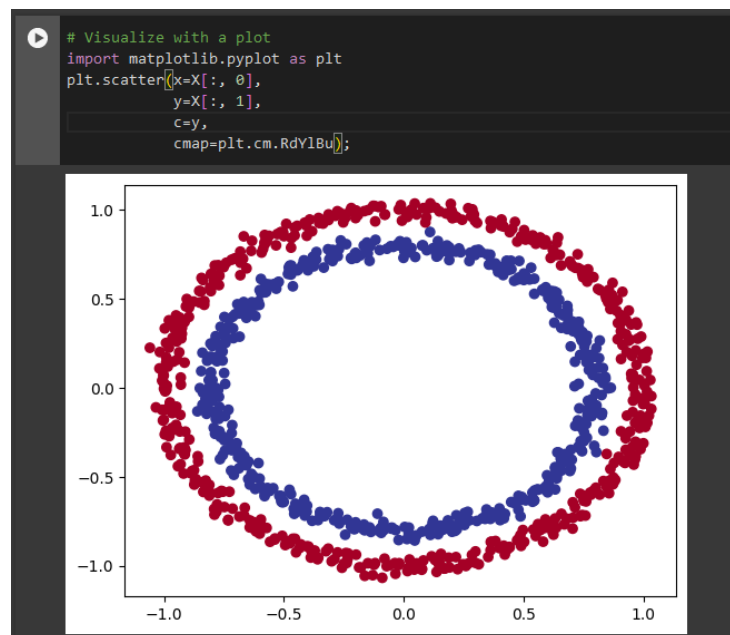
# Create circles
X, y = make_circles(n_samples,
                    noise=0.03, # a little b
                    random_state=42) # keep
```

Blok kode di atas menggunakan fungsi `make_circles` dari modul `sklearn.datasets` untuk membuat dataset yang berisi dua lingkaran yang bersilangan. mengimpor fungsi `make_circles` dari modul `sklearn.datasets`. Fungsi ini digunakan untuk membuat dataset berisi dua lingkaran yang bersilangan, dan menentukan jumlah sampel yang ingin dibuat dalam dataset, yaitu 1000 sampel.

```
# Make DataFrame of circle data
import pandas as pd
circles = pd.DataFrame({"X1": X[:, 0],
                        "X2": X[:, 1],
                        "label": y})
circles.head(10)
```

	X1	X2	label
0	0.754246	0.231481	1
1	-0.756159	0.153259	1
2	-0.815392	0.173282	1
3	-0.393731	0.692883	1
4	0.442208	-0.896723	0
5	-0.479646	0.676435	1
6	-0.013648	0.803349	1
7	0.771513	0.147760	1
8	-0.169322	-0.793456	1
9	-0.121486	1.021509	0

DataFrame yang berisi koordinat X1 dan X2 dari setiap titik dalam dataset lingkaran, beserta label label yang menunjukkan apakah titik tersebut berada di dalam atau di luar lingkaran yang bersilangan.



menggunakan metode `value_counts()` pada kolom label dari DataFrame `circles` untuk menghitung frekuensi kemunculan setiap nilai label. `circles.label.value_counts()`: Metode ini

menghitung jumlah kemunculan setiap nilai dalam kolom label. Dalam konteks ini, label 0 dan 1 digunakan untuk menunjukkan apakah suatu titik berada di dalam atau di luar lingkaran.

```
# View the first 5 outputs of the forward pass
y_logits = model_0(X_test.to(device))[:5]
y_logits

tensor([[0.4928],
        [0.5514],
        [0.4518],
        [0.5039],
        [0.6307]], grad_fn=<SliceBackward0>)

# Use sigmoid on model logits
y_pred_probs = torch.sigmoid(y_logits)
y_pred_probs

tensor([[0.6208],
        [0.6345],
        [0.6111],
        [0.6234],
        [0.6527]], grad_fn=<SigmoidBackward0>)
```

Blok kode di atas menampilkan lima output pertama dari hasil operasi forward pass pada data pengujian (`X_test`) menggunakan model (`model_0`). tensor yang berisi lima output pertama dari operasi forward pass pada data pengujian. Ini mungkin berupa nilai-nilai logits atau nilai-nilai aktivasi, tergantung pada arsitektur dan lapisan-lapisan model yang digunakan.

`torch.sigmoid(y_logits)`: Ini adalah fungsi sigmoid yang diterapkan pada logits (`y_logits`). Fungsi sigmoid mengubah nilai-nilai logits menjadi rentang antara 0 dan 1, yang dapat diinterpretasikan sebagai probabilitas. Hasilnya adalah tensor yang berisi probabilitas prediksi untuk lima output pertama dari hasil operasi forward pass pada data pengujian. Probabilitas ini mengindikasikan seberapa yakin model pada setiap kelas (misalnya, kelas 0 atau kelas 1).

```
# Find the predicted labels (round the prediction probabilities)
y_preds = torch.round(y_pred_probs)

# In full
y_pred_labels = torch.round(torch.sigmoid(model_0(X_test.to(device))[:5]))

# Check for equality
print(torch.eq(y_preds.squeeze(), y_pred_labels.squeeze()))

# Get rid of extra dimension
y_preds.squeeze_()

tensor([True, True, True, True, True])
tensor([1., 1., 1., 1., 1.], grad_fn=<SqueezeBackward0>)
```

kode di atas digunakan untuk menghasilkan prediksi label biner dari probabilitas prediksi dengan membulatkan ke nilai terdekat (0 atau 1). Hasilnya adalah perbandingan kesamaan antara dua metode untuk mendapatkan prediksi label dan tensor prediksi akhir yang sudah dihilangkan dimensi ekstra.


```

# Set the number of epochs
epochs = 100

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

# Build training and evaluation loop
for epoch in range(epochs):
    ### Training
    model_0.train()

    # 1. Forward pass (model outputs raw logits)
    y_logits = model_0(X_train).squeeze() # squeeze to remove extra '1' dimensions, this won't work unless model and data are on same device
    y_pred = torch.round(torch.sigmoid(y_logits)) # turn logits -> pred probs -> pred labels

    # 2. Calculate loss/accuracy
    # loss = loss_fn(torch.sigmoid(y_logits), # Using nn.BCELoss you need torch.sigmoid()
    # y_train)
    loss = loss_fn(y_logits, # Using nn.BCEWithLogitsLoss works with raw logits
    y_train)
    acc = accuracy_fn(y_true=y_train,
    y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

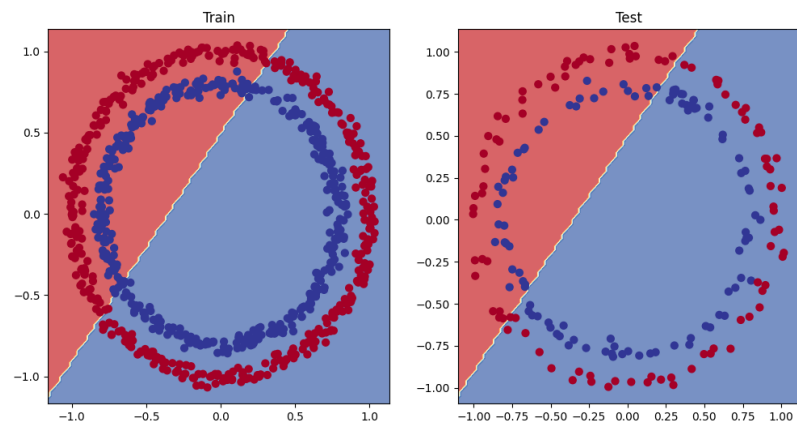
    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_0.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_logits = model_0(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))
        # 2. Calculate loss/accuracy
        test_loss = loss_fn(test_logits,
        y_test)
        test_acc = accuracy_fn(y_true=y_test,
        y_pred=test_pred)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% | Test loss: {test_loss:.5f}, Test acc: {test_acc:.2f}%")

```

kode di atas merupakan implementasi dari siklus pelatihan (training loop) dan evaluasi (testing loop) untuk model yang menggunakan fungsi kerugian (loss function) dan pengoptimal (optimizer). Hasilnya adalah pelatihan dan evaluasi model dengan mencetak loss, akurasi, test loss, dan test accuracy pada setiap 10 epochs. Hal ini membantu untuk melihat bagaimana model memperbaiki performanya selama proses pelatihan.



Gambar diatas menjelaskan program sedang mencoba untuk memisahkan titik merah dan biru menggunakan garis lurus. Itu menjelaskan akurasi 50%. Karena data kita berbentuk lingkaran, menggambar garis lurus paling baik dapat memotongnya di tengah. Dalam istilah pembelajaran mesin, model kami kurang optimal, artinya model kami tidak mempelajari pola prediktif dari data.

```

torch.manual_seed(42)

epochs = 1000 # Train for longer

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

for epoch in range(epochs):
    ### Training
    # 1. Forward pass
    y_logits = model_1(X_train).squeeze()
    y_pred = torch.round(torch.sigmoid(y_logits)) # logits -> pred

    # 2. Calculate loss/accuracy
    loss = loss_fn(y_logits, y_train)
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_1.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_logits = model_1(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))
        # 2. Caculate loss/accuracy
        test_loss = loss_fn(test_logits,
                             y_test)
        test_acc = accuracy_fn(y_true=y_test,
                               y_pred=test_pred)

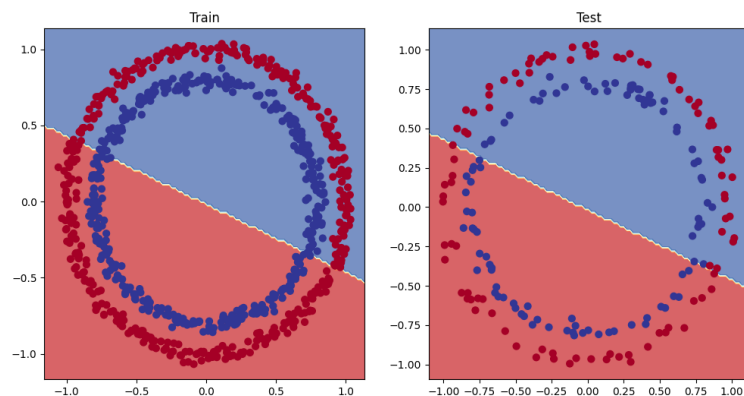
```

```

Epoch: 0 | Loss: 0.69396, Accuracy: 50.88% | Test loss: 0.69261, Test acc: 51.00%
Epoch: 100 | Loss: 0.69305, Accuracy: 50.38% | Test loss: 0.69379, Test acc: 48.00%
Epoch: 200 | Loss: 0.69299, Accuracy: 51.12% | Test loss: 0.69437, Test acc: 46.00%
Epoch: 300 | Loss: 0.69298, Accuracy: 51.62% | Test loss: 0.69458, Test acc: 45.00%
Epoch: 400 | Loss: 0.69298, Accuracy: 51.12% | Test loss: 0.69465, Test acc: 46.00%
Epoch: 500 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69467, Test acc: 46.00%
Epoch: 600 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 46.00%
Epoch: 700 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 46.00%
Epoch: 800 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 46.00%
Epoch: 900 | Loss: 0.69298, Accuracy: 51.00% | Test loss: 0.69468, Test acc: 46.00%

```

kode di atas adalah implementasi siklus pelatihan dan evaluasi yang lebih panjang untuk model_1 dengan jumlah epochs yang diperpanjang. pelatihan dan evaluasi model yang dilakukan selama 1000 epochs, dengan mencetak loss, akurasi, test loss, dan test accuracy setiap 100 epochs.



Model masih menggambar garis lurus antara titik merah dan biru. Jika model kita menggambar garis lurus, ada kemungkinan model tersebut dapat memodelkan data linier.

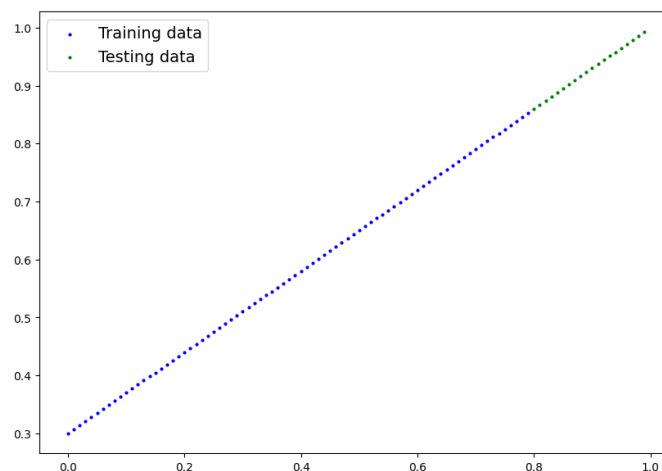
```
# Create some data (same as notebook 01)
weight = 0.7
bias = 0.3
start = 0
end = 1
step = 0.01

# Create data
X_regression = torch.arange(start, end, step).unsqueeze(dim=1)
y_regression = weight * X_regression + bias # linear regression

# Check the data
print(len(X_regression))
X_regression[:5], y_regression[:5]
```

```
100
(tensor([[0.0000],
         [0.0100],
         [0.0200],
         [0.0300],
         [0.0400]]),
 tensor([[0.3000],
         [0.3070],
         [0.3140],
         [0.3210],
         [0.3280]]))
```

kode di atas membuat data untuk masalah regresi linear, dengan parameter weight (bobot) dan bias tertentu. Data ini dibuat dengan menggunakan rumus regresi linear $y = \text{weight} * X + \text{bias}$. tensor `X_regression` yang berisi nilai-nilai dalam rentang $[0, 1)$ dengan langkah 0.01, dan tensor `y_regression` yang dihasilkan dari regresi linear dengan bobot 0.7 dan bias 0.3.



Garis data pelatihan dimulai di titik (0,0) dan naik ke titik (1,1). Hal ini menunjukkan bahwa model pembelajaran mesin dapat menghasilkan hasil yang sempurna pada data pelatihan. Namun, garis data uji tidak mencapai titik (1,1). Hal ini menunjukkan bahwa model pembelajaran mesin tidak dapat menghasilkan hasil yang sempurna pada data uji.

```
# Loss and optimizer
loss_fn = nn.L1Loss()
optimizer = torch.optim.SGD(model_2.parameters(), lr=0.1)
```

Blok kode di atas mendefinisikan fungsi loss dan optimizer untuk model regresi linear (model_2). Dengan konfigurasi ini, model akan di-training menggunakan L1 Loss dan dioptimalkan dengan SGD dengan learning rate 0.1.

```
# Train the model
torch.manual_seed(42)

# Set the number of epochs
epochs = 1000

# Put data to target device
X_train_regression, y_train_regression = X_train_regression.to(device), y_train_regression.to(device)
X_test_regression, y_test_regression = X_test_regression.to(device), y_test_regression.to(device)

for epoch in range(epochs):
    ### Training
    # 1. Forward pass
    y_pred = model_2(X_train_regression)

    # 2. Calculate loss (no accuracy since it's a regression problem, not classification)
    loss = loss_fn(y_pred, y_train_regression)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

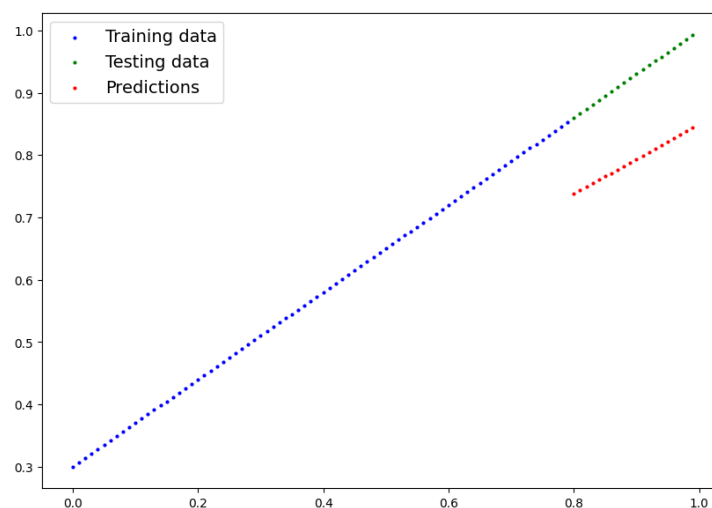
    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_2.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_pred = model_2(X_test_regression)
        # 2. Calculate the loss
        test_loss = loss_fn(test_pred, y_test_regression)

    # Print out what's happening
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Train loss: {loss:.5f}, Test loss: {test_loss:.5f}")
```

kode di atas melatih model regresi linear (model_2) untuk memprediksi nilai kontinu.



Grafik ini menunjukkan bahwa data pelatihan dan data pengujian memiliki hubungan yang positif. Artinya, semakin banyak data pelatihan yang digunakan, semakin baik kinerja model pada data pengujian. Grafik ini juga menunjukkan bahwa ada perbedaan antara data pelatihan dan data pengujian. Hal ini disebabkan karena data pengujian biasanya lebih beragam daripada data pelatihan.

```
# Fit the model
torch.manual_seed(42)
epochs = 1000

# Put all data on target device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

for epoch in range(epochs):
    # 1. Forward pass
    y_logits = model_3(X_train).squeeze()
    y_pred = torch.round(torch.sigmoid(y_logits)) # logits

    # 2. Calculate loss and accuracy
    loss = loss_fn(y_logits, y_train) # BCEWithLogitsLoss
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    optimizer.step()
```

Epoch: 0	Loss: 0.69295, Accuracy: 50.00%	Test Loss: 0.69319, Test Accuracy: 50.00%
Epoch: 100	Loss: 0.69115, Accuracy: 52.88%	Test Loss: 0.69102, Test Accuracy: 52.50%
Epoch: 200	Loss: 0.68977, Accuracy: 53.37%	Test Loss: 0.68940, Test Accuracy: 55.00%
Epoch: 300	Loss: 0.68795, Accuracy: 53.00%	Test Loss: 0.68723, Test Accuracy: 56.00%
Epoch: 400	Loss: 0.68517, Accuracy: 52.75%	Test Loss: 0.68411, Test Accuracy: 56.50%
Epoch: 500	Loss: 0.68102, Accuracy: 52.75%	Test Loss: 0.67941, Test Accuracy: 56.50%
Epoch: 600	Loss: 0.67515, Accuracy: 54.50%	Test Loss: 0.67285, Test Accuracy: 56.00%
Epoch: 700	Loss: 0.66659, Accuracy: 58.38%	Test Loss: 0.66322, Test Accuracy: 59.00%
Epoch: 800	Loss: 0.65160, Accuracy: 64.00%	Test Loss: 0.64757, Test Accuracy: 67.50%
Epoch: 900	Loss: 0.62362, Accuracy: 74.00%	Test Loss: 0.62145, Test Accuracy: 79.00%

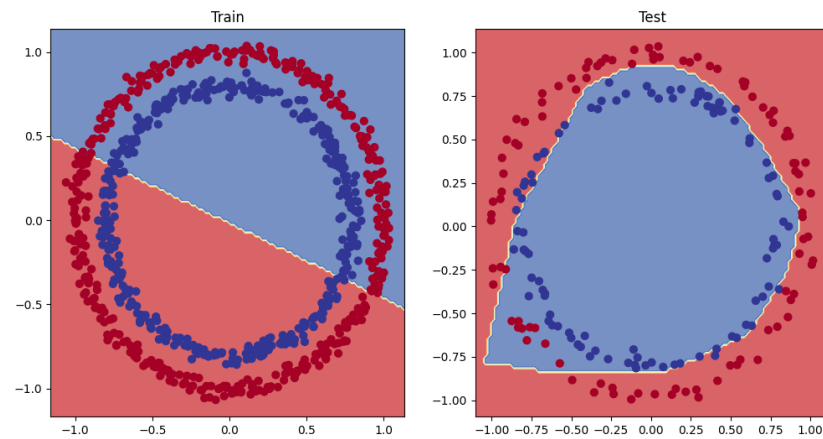
Program di atas melatih model klasifikasi biner (model_3) untuk memprediksi label biner (0 atau 1). Mencetak loss dan akurasi pada data pelatihan serta loss dan akurasi pada data pengujian setiap 100 epoch.

```
# Make predictions
model_3.eval()
with torch.inference_mode():
    y_preds = torch.round(torch.sigmoid(model_3(X_test))).squeeze()
    y_preds[:10], y[:10] # want preds in same format as truth labels

(tensor([1., 0., 1., 0., 0., 1., 0., 0., 1., 0.]),
 tensor([1., 1., 1., 1., 0., 1., 1., 1., 1., 0.]))
```

Dalam bagian ini, model yang telah dilatih (model_3) diubah ke mode evaluasi, dan dengan menggunakan mode inferensi, prediksi dibuat pada data pengujian (X_test). Fungsi sigmoid digunakan untuk mengubah logit menjadi probabilitas, dan kemudian dilakukan pembulatan ke

label biner (0 atau 1) dengan fungsi `torch.round()`. Hasil prediksi (`y_preds`) dan label sebenarnya (`y`) untuk 10 sampel pertama kemudian dicetak.



```
# Import dependencies
import torch
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split

# Set the hyperparameters for data creation
NUM_CLASSES = 4
NUM_FEATURES = 2
RANDOM_SEED = 42

# 1. Create multi-class data
X_blob, y_blob = make_blobs(n_samples=1000,
                             n_features=NUM_FEATURES, # X features
                             centers=NUM_CLASSES, # y labels
                             cluster_std=1.5, # give the clusters a little shake up (try changing this t
                             random_state=RANDOM_SEED
)

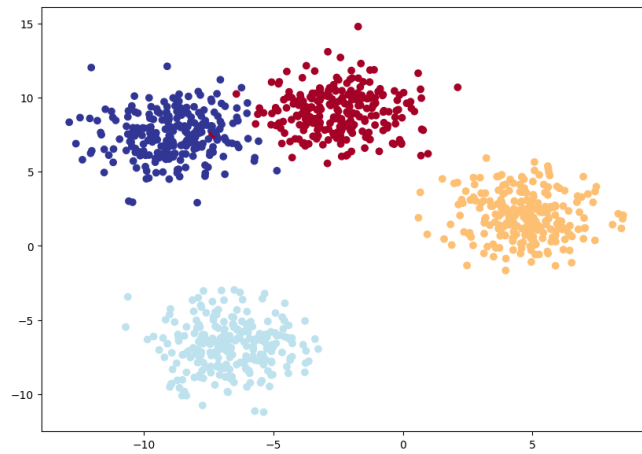
# 2. Turn data into tensors
X_blob = torch.from_numpy(X_blob).type(torch.float)
y_blob = torch.from_numpy(y_blob).type(torch.LongTensor)
print(X_blob[:5], y_blob[:5])

# 3. Split into train and test sets
X_blob_train, X_blob_test, y_blob_train, y_blob_test = train_test_split(X_blob,
                                y_blob,
                                test_size=0.2,
                                random_state=RANDOM_SEED
)

# 4. Plot data
plt.figure(figsize=(10, 7))
plt.scatter(X_blob[:, 0], X_blob[:, 1], c=y_blob, cmap=plt.cm.RdYlBu);
```

Pada bagian ini, data digenerate menggunakan fungsi `make_blobs` dari library scikit-learn. Data yang dihasilkan memiliki 2 fitur dan terbagi menjadi 4 kelas (ditentukan oleh parameter `NUM_CLASSES`). Data yang dihasilkan kemudian diubah menjadi tensor PyTorch menggunakan `torch.from_numpy()` dan tipe data tensor diatur menjadi float untuk fitur (`X_blob`) dan LongTensor untuk label kelas (`y_blob`). Selanjutnya, data dibagi menjadi set pelatihan dan pengujian

menggunakan fungsi `train_test_split` dari `scikit-learn`. Hasilnya ditampilkan dalam sebuah diagram pencar menggunakan `matplotlib`.



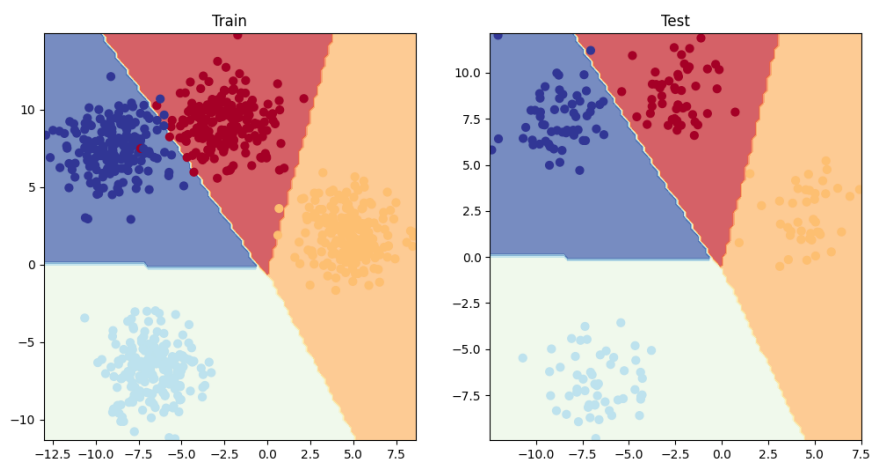
```
# Turn predicted logits in prediction probabilities
y_pred_probs = torch.softmax(y_logits, dim=1)

# Turn prediction probabilities into prediction labels
y_preds = y_pred_probs.argmax(dim=1)

# Compare first 10 model preds and test labels
print(f"Predictions: {y_preds[:10]}\nLabels: {y_blob_test[:10]}")
print(f"Test accuracy: {accuracy_fn(y_true=y_blob_test, y_pred=y_preds)}%")

Predictions: tensor([1, 3, 2, 1, 0, 3, 2, 0, 2, 0])
Labels: tensor([1, 3, 2, 1, 0, 3, 2, 0, 2, 0])
Test accuracy: 99.5%
```

Dalam bagian ini, logit yang dihasilkan dari model (`y_logits`) diubah menjadi probabilitas prediksi menggunakan fungsi `softmax`. Kemudian, probabilitas prediksi diubah menjadi label prediksi dengan mengambil `argmax` dari dimensi kedua (dimensi yang merepresentasikan kelas). Hasilnya dibandingkan dengan label sebenarnya dari data pengujian (`y_blob_test`). Akurasi dihitung menggunakan fungsi `accuracy_fn` dan dicetak.



❖ CHAPTER 03 PyTorch Computer Vision

```
# Setup training data
train_data = datasets.FashionMNIST(
    root="data", # where to download data
    train=True, # get training data
    download=True, # download data if it d
    transform=ToTensor(), # images come as
    target_transform=None # you can transi
)

# Setup testing data
test_data = datasets.FashionMNIST(
    root="data",
    train=False, # get test data
    download=True,
    transform=ToTensor()
```

Program di atas merupakan contoh kode Python untuk mempersiapkan data pelatihan (training data) dan data pengujian (testing data) untuk model pembelajaran mesin menggunakan dataset FashionMNIST. Dataset ini biasanya digunakan untuk tugas pengenalan gambar di bidang mode. Dengan cara ini, dataset FashionMNIST diunduh (jika belum ada), dan gambar-gambar diubah menjadi tensor PyTorch menggunakan transformasi `ToTensor()`. Data pelatihan dan pengujian kemudian siap untuk digunakan dalam melatih dan menguji model pembelajaran mesin.

```
# See first training sample
image, label = train_data[0]
image, label

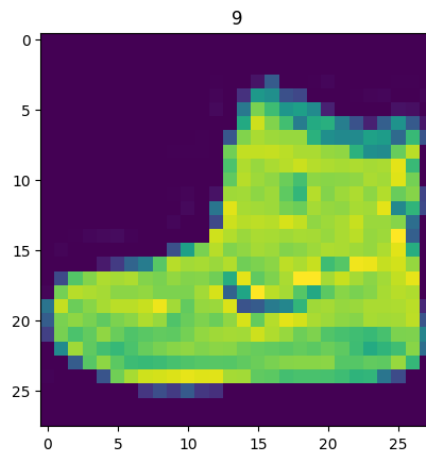
(tensor([[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.0000, 0.0510,
0.2863, 0.0000, 0.0000, 0.0039, 0.0157, 0.0000, 0.0000, 0.0000,
0.0000, 0.0039, 0.0039, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0118, 0.0000, 0.1412, 0.5333,
0.4980, 0.2431, 0.2118, 0.0000, 0.0000, 0.0000, 0.0039, 0.0118,
0.0157, 0.0000, 0.0000, 0.0118],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0235, 0.0000, 0.4000, 0.8000,
0.6902, 0.5255, 0.5647, 0.4824, 0.0902, 0.0000, 0.0000, 0.0000,
0.0000, 0.0471, 0.0392, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.6078, 0.9255,
0.8118, 0.6980, 0.4196, 0.6118, 0.6314, 0.4275, 0.2510, 0.0902,
0.3020, 0.5098, 0.2824, 0.0588],
```

Kode di atas digunakan untuk mengambil dan menampilkan sampel pertama dari data pelatihan (`train_data`). `image, label = train_data[0]`: Mengakses elemen pertama dari dataset pelatihan. Setiap elemen dataset FashionMNIST terdiri dari sebuah gambar dan label yang sesuai. Oleh karena itu, `image` akan berisi tensor representasi gambar, dan `label` akan berisi label kelas yang sesuai.


```

import matplotlib.pyplot as plt
image, label = train_data[0]
print(f"Image shape: {image.shape}")
plt.imshow(image.squeeze()) # image shape
plt.title(label);

```

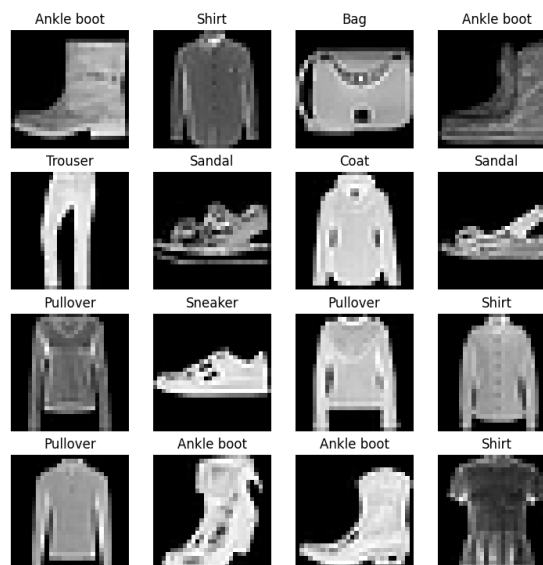


menampilkan gambar pertama dari dataset pelatihan bersama dengan labelnya menggunakan Matplotlib. menggunakan `squeeze()` untuk menghapus dimensi saluran warna yang tidak diperlukan, karena gambar FashionMNIST adalah gambar grayscale dan hanya memiliki satu saluran warna. Fungsi ini menghasilkan visualisasi gambar menggunakan Matplotlib.

```

# Plot more images
torch.manual_seed(42)
fig = plt.figure(figsize=(9, 9))
rows, cols = 4, 4
for i in range(1, rows * cols + 1):
    random_idx = torch.randint(0, len(train_data), size=[1]).item()
    img, label = train_data[random_idx]
    fig.add_subplot(rows, cols, i)
    plt.imshow(img.squeeze(), cmap="gray")
    plt.title(class_names[label])
    plt.axis(False);

```



Kode berikut digunakan untuk membuat tampilan gambar acak dari dataset pelatihan dalam bentuk matriks 4x4. Dengan menggunakan kode ini, dapat melihat matriks 4x4 dari gambar acak dari dataset FashionMNIST bersama dengan label kelasnya dalam tampilan visual.

```
from torch.utils.data import DataLoader

# Setup the batch size hyperparameter
BATCH_SIZE = 32

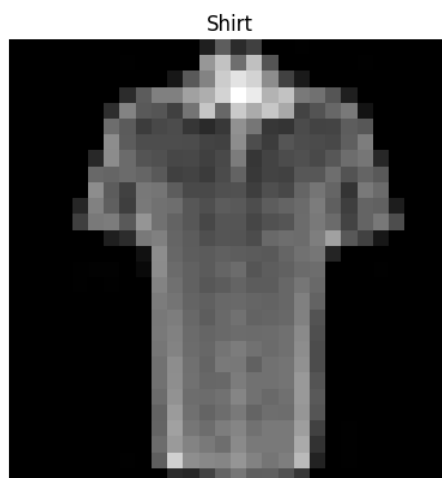
# Turn datasets into iterables (batches)
train_dataloader = DataLoader(train_data, # dataset to turn into iterable
                              batch_size=BATCH_SIZE, # how many samples per batch?
                              shuffle=True # shuffle data every epoch?
                              )

test_dataloader = DataLoader(test_data,
                             batch_size=BATCH_SIZE,
                             shuffle=False # don't necessarily have to shuffle the testing data
                             )

# Let's check out what we've created
print(f"Dataloaders: {train_dataloader, test_dataloader}")
print(f"Length of train dataloader: {len(train_dataloader)} batches of {BATCH_SIZE}")
print(f"Length of test dataloader: {len(test_dataloader)} batches of {BATCH_SIZE}")
```

Kode di atas menggunakan modul DataLoader dari PyTorch untuk mengonversi dataset pelatihan dan pengujian menjadi iterables (batches) dengan ukuran batch yang ditentukan. DataLoader digunakan untuk membuat iterable dari dataset PyTorch, yang memungkinkan untuk mengambil batch data secara efisien selama pelatihan model.

```
# Show a sample
torch.manual_seed(42)
random_idx = torch.randint(0, len(train_features_batch), size=[1]).item()
img, label = train_features_batch[random_idx], train_labels_batch[random_idx]
plt.imshow(img.squeeze(), cmap="gray")
plt.title(class_names[label])
plt.axis("Off");
print(f"Image size: {img.shape}")
print(f"Label: {label}, label size: {label.shape}")
```



Program ini berfokus pada menampilkan sampel acak dari batch pelatihan yang dihasilkan oleh DataLoader.

```

# Import tqdm for progress bar
from tqdm.auto import tqdm

# Set the seed and start the timer
torch.manual_seed(42)
train_time_start_on_cpu = timer()

# Set the number of epochs (we'll keep this small for faster training times)
epochs = 3

# Create training and testing loop
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n-----")
    ### Training
    train_loss = 0
    # Add a loop to loop through training batches
    for batch, (X, y) in enumerate(train_dataloader):
        model_0.train()
        # 1. Forward pass
        y_pred = model_0(X)

        # 2. Calculate loss (per batch)
        loss = loss_fn(y_pred, y)
        train_loss += loss # accumulatively add up the loss per epoch

        # 3. Optimizer zero grad
        optimizer.zero_grad()

        # 4. Loss backward
        loss.backward()

        # 5. Optimizer step
        optimizer.step()

    # Print out how many samples have been seen
    if batch % 400 == 0:
        print(f"Looked at {batch * len(X)}/{len(train_dataloader.dataset)} samples")

```

```

100% 3/3 [00:24<00:00, 8.16s/it]
Epoch: 0
-----
Looked at 0/60000 samples
Looked at 12800/60000 samples
Looked at 25600/60000 samples
Looked at 38400/60000 samples
Looked at 51200/60000 samples

Train loss: 0.59039 | Test loss: 0.50954, Test acc: 82.04%

Epoch: 1
-----
Looked at 0/60000 samples
Looked at 12800/60000 samples
Looked at 25600/60000 samples
Looked at 38400/60000 samples
Looked at 51200/60000 samples

Train loss: 0.47633 | Test loss: 0.47989, Test acc: 83.20%

Epoch: 2
-----
Looked at 0/60000 samples
Looked at 12800/60000 samples
Looked at 25600/60000 samples
Looked at 38400/60000 samples
Looked at 51200/60000 samples

Train loss: 0.45503 | Test loss: 0.47664, Test acc: 83.43%

Train time on cpu: 24.567 seconds

```

Program ini mencakup proses pelatihan dan evaluasi model menggunakan PyTorch, dengan menampilkan informasi mengenai loss dan akurasi pada setiap epoch. Selain itu, digunakan juga progress bar (tqdm) untuk memberikan indikasi visual tentang kemajuan pelatihan. `model_0.train()`: Mengatur model dalam mode pelatihan. Forward pass, perhitungan loss, backpropagation, dan optimasi model.

```

# Plot predictions
plt.figure(figsize=(9, 9))
nrows = 3
ncols = 3
for i, sample in enumerate(test_samples):
    # Create a subplot
    plt.subplot(nrows, ncols, i+1)

    # Plot the target image
    plt.imshow(sample.squeeze(), cmap="gray")

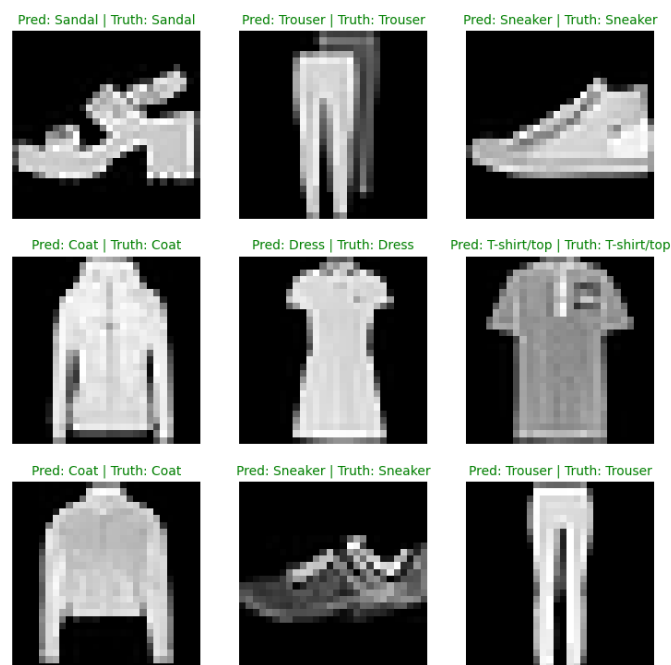
    # Find the prediction label (in text form, e.g. "Sandal")
    pred_label = class_names[pred_classes[i]]

    # Get the truth label (in text form, e.g. "T-shirt")
    truth_label = class_names[test_labels[i]]

    # Create the title text of the plot
    title_text = f"Pred: {pred_label} | Truth: {truth_label}"

    # Check for equality and change title colour accordingly
    if pred_label == truth_label:
        plt.title(title_text, fontsize=10, c="g") # green text if
    else:
        plt.title(title_text, fontsize=10, c="r") # red text if
    plt.axis(False);

```



Program ini menghasilkan matriks subplot yang menunjukkan beberapa prediksi model pada sampel-sampel pengujian. Setiap subplot berisi gambar target, label prediksi, dan label sebenarnya. Judul setiap subplot ditampilkan dengan warna hijau jika prediksi benar dan warna merah jika prediksi salah.

```
# Import tqdm for progress bar
from tqdm.auto import tqdm

# 1. Make predictions with trained model
y_preds = []
model_2.eval()
with torch.inference_mode():
    for X, y in tqdm(test_dataloader, desc="Making predictions"):
        # Send data and targets to target device
        X, y = X.to(device), y.to(device)
        # Do the forward pass
        y_logits = model_2(X)
        # Turn predictions from logits -> prediction probabilities -> predictions labels
        y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1) # note: perform softmax on the logits
        # Put predictions on CPU for evaluation
        y_preds.append(y_pred.cpu())
# Concatenate list of predictions into a tensor
y_pred_tensor = torch.cat(y_preds)
```

Making predictions: 100% 313/313 [00.03<00.00, 91.49it/s]

Dengan menggunakan kode ini, kita mendapatkan tensor `y_pred_tensor` yang berisi prediksi model untuk seluruh dataset pengujian. Ini dapat digunakan untuk evaluasi lebih lanjut seperti perhitungan akurasi atau matriks kebingungan (confusion matrix).

```
# See if torchmetrics exists, if not, install it
try:
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
    assert int(mlxtend.__version__.split(".")[1]) >= 19, "mlxtend version should be at least 19"
except:
    !pip install -q torchmetrics -U mlxtend # <- Note: If you're using Google Colab, you need to use !pip
    import torchmetrics, mlxtend
    print(f"mlxtend version: {mlxtend.__version__}")
```

806.1/806.1 kB 11.3 MB/s eta 0:00:00
1.4/1.4 MB 21.0 MB/s eta 0:00:00
mlxtend version: 0.23.0

Dengan cara ini, program memastikan bahwa modul `torchmetrics` dan `mlxtend` tersedia dan dalam versi yang diperlukan. Jika modul tersebut tidak ada, maka program akan menginstalnya sebelum melanjutkan.

```
from torchmetrics import ConfusionMatrix
from mlxtend.plotting import plot_confusion_matrix

# 2. Setup confusion matrix instance and compare predictions to targets
confmat = ConfusionMatrix(num_classes=len(class_names), task='multiclass')
confmat_tensor = confmat(preds=y_pred_tensor,
                        target=test_data.targets)

# 3. Plot the confusion matrix
fig, ax = plot_confusion_matrix(
    conf_mat=confmat_tensor.numpy(), # matplotlib likes working with NumPy
    class_names=class_names, # turn the row and column labels into class names
    figsize=(10, 7)
);
```

T-shirt/top	829	1	9	28	6	1	120	0	6	0
Trouser	3	971	0	18	2	0	5	0	1	0
Pullover	13	1	726	10	153	0	95	0	2	0
Dress	15	5	7	919	22	0	31	0	1	0
Coat	0	0	34	45	830	0	91	0	0	0
Sandal	0	0	0	2	0	983	0	11	1	3
Shirt	125	0	58	37	75	0	692	0	13	0
Sneaker	0	0	0	0	0	37	0	927	0	36
Bag	3	1	3	9	2	3	12	5	962	0
Ankle boot	0	0	0	1	0	6	1	29	1	962

Kode di atas digunakan untuk membuat dan memvisualisasikan matriks kebingungan (confusion matrix) menggunakan modul torchmetrics dan mlxtend. Dengan cara ini, program menghasilkan dan memvisualisasikan matriks kebingungan yang memberikan wawasan tentang performa model pada setiap kelas. Matriks kebingungan ini membandingkan prediksi model dengan label sebenarnya pada dataset pengujian.

```
# Evaluate loaded model
torch.manual_seed(42)

loaded_model_2_results = eval_model(
    model=loaded_model_2,
    data_loader=test_dataloader,
    loss_fn=loss_fn,
    accuracy_fn=accuracy_fn
)

loaded_model_2_results

{'model_name': 'FashionMNISTModelV2',
 'model_loss': 0.3212365508079529,
 'model_acc': 88.01916932907348}
```

pemanggilan fungsi `eval_model` untuk mengevaluasi model yang telah dimuat (`loaded_model_2`). Namun, kode untuk fungsi `eval_model` sendiri tidak disertakan dalam pertanyaan Anda. Output tersebut merupakan hasil evaluasi model pada dataset pengujian (test set) dengan beberapa metrik performa. Hasil evaluasi ini memberikan gambaran tentang seberapa baik model "FashionMNISTModelV2" berperforma pada dataset pengujian, diukur dengan loss dan akurasi. Semakin rendah loss dan semakin tinggi akurasi, semakin baik performa model tersebut.