Nama : Ivan Fernanda Prayoga

NIM : 1103204035

Kelas : TK44G4

UAS Machine Learning

PyTorch Fundamentals

PyTorch adalah kerangka kerja (framework) machine learning dan deep learning sumber terbuka (open source). PyTorch memungkinkan untuk memanipulasi dan memproses data serta menulis algoritma machine learning menggunakan kode Python. PyTorch dapat digunakan untuk berbagai keperluan seperti pengolahan data, pengembangan algoritma machine learning, dan pengembangan model deep learning. Dengan PyTorch, Kita dapat melakukan tugas seperti pengenalan gambar, pemrosesan bahasa alami, dan pengenalan suara. Banyak perusahaan teknologi terkemuka di dunia seperti Meta (Facebook), Tesla, dan Microsoft, serta perusahaan riset kecerdasan buatan seperti OpenAI menggunakan PyTorch untuk melakukan penelitian dan mengimplementasikan machine learning dalam produk-produk mereka. PyTorch telah menjadi salah satu pilihan populer di kalangan praktisi dan peneliti machine learning.

Pada PyTorch Fundamentals ini akan membahas tentang :

- Introduction to tensors: Tensor adalah komponen dasar dalam machine learning dan deep learning.
- Creating tensors: Tensor dapat merepresentasikan berbagai jenis data seperti gambar, kata-kata, dan tabel angka.
- Getting information from tensors: Jika Kita dapat memasukkan informasi ke dalam tensor, Kita juga akan ingin mengambil informasinya.
- Manipulating tensors: Algoritma machine learning, seperti jaringan saraf (neural networks), melibatkan manipulasi tensor dengan cara yang berbeda, seperti penambahan, perkalian, dan penggabungan.
- Dealing with tensor shapes: Salah satu masalah umum dalam machine learning adalah menangani ketidakcocokan bentuk (shape) tensor, misalnya mencoba menggabungkan tensor dengan bentuk yang salah.
- Indexing on tensors: Jika Kita pernah menggunakan indeks pada daftar Python atau array NumPy, konsepnya hampir sama dengan tensor, hanya saja tensor dapat memiliki dimensi yang lebih banyak.
- Mixing PyTorch tensors and NumPy: PyTorch menggunakan tensor (torch.Tensor), sedangkan NumPy menggunakan array (np.ndarray). Kadang-kadang Kita ingin menggabungkan keduanya.
- Reproducibility: Machine learning bersifat eksperimental dan karena menggunakan banyak keacakan, kadang-kadang Kita ingin mengontrol agar keacakan tersebut tidak terlalu acak.
- Running tensors on GPU: GPU (Graphics Processing Unit) dapat mempercepat kode Kita. PyTorch memudahkan Kita untuk menjalankan kode pada GPU.

Importing PyTorch

```
import torch
torch.__version__
```

```
'2.1.0+cu121'
```

Kode program menampilkan output versi PyTorch.

Creating tensors

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

```
tensor(7)
```

```
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])
```

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

```
1
```

Kode program melakukan :

- Tensor skalar dengan nilai 7 dibuat menggunakan fungsi torch.tensor(), lalu menampilkan outputnya.
- scalar.ndim digunakan untuk mendapatkan jumlah dimensi tensor. Namun, karena scalar adalah tensor skalar, artinya memiliki dimensi tunggal, maka hasilnya adalah 0.
- scalar.item() digunakan untuk mengambil nilai skalar dalam tensor. Namun, metode .item() hanya berfungsi dengan tensor yang hanya memiliki satu elemen. Dalam kasus ini, kita dapat menggunakan .item() untuk mendapatkan nilai skalar 7.
- Sebuah tensor vektor dengan nilai [7, 7] dibuat menggunakan fungsi torch.tensor(). Lalu menampilkan output dari variabel vector.
- vector.ndim digunakan untuk mendapatkan jumlah dimensi tensor. Tensor vektor memiliki satu dimensi, karena hanya memiliki satu sumbu (axis), sehingga hasilnya adalah 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])
```

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

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

Kode program melakukan :

- vector.shape digunakan untuk memeriksa bentuk (shape) dari tensor vektor. Dalam kasus ini, bentuknya adalah (2,), yang berarti tensor vektor memiliki 2 elemen.
- Sebuah tensor matriks dengan nilai [[7, 8], [9, 10]] dibuat menggunakan fungsi torch.tensor(). Lalu menampilkan output dari variabel MATRIX.
- MATRIX.ndim digunakan untuk memeriksa jumlah dimensi dari tensor matriks. Tensor matriks memiliki dua dimensi, karena memiliki dua sumbu (axis), sehingga hasilnya adalah 2.
- MATRIX.shape digunakan untuk memeriksa bentuk (shape) dari tensor matriks. Dalam kasus ini, bentuknya adalah (2, 2), yang berarti tensor matriks memiliki 2 baris dan 2 kolom.

- sebuah tensor tiga dimensi dengan nilai [[[1, 2, 3], [3, 6, 9], [2, 4, 5]]] dibuat menggunakan fungsi torch.tensor(). Lalu menampilkan output dari variabel TENSOR

```
# Check number of dimensions for TENSOR
TENSOR.ndim
```

```
3
```

```
# Check shape of TENSOR
TENSOR.shape
```

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

Kode program melakukan :

- TENSOR.ndim digunakan untuk memeriksa jumlah dimensi dari tensor TENSOR. Tensor TENSOR memiliki tiga dimensi, karena memiliki tiga sumbu (axis), sehingga hasilnya adalah 3.
- TENSOR.shape digunakan untuk memeriksa bentuk (shape) dari tensor TENSOR. Dalam kasus ini, bentuknya adalah (1, 3, 3), yang berarti tensor TENSOR memiliki 1 lapisan, dengan setiap lapisan memiliki 3 baris dan 3 kolom.

Random Tensors

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

```
(tensor([[0.3981, 0.1118, 0.9076, 0.3175],
         [0.9322, 0.0028, 0.7621, 0.7791],
         [0.3913, 0.8103, 0.7264, 0.1456]]),
 torch.float32)
```

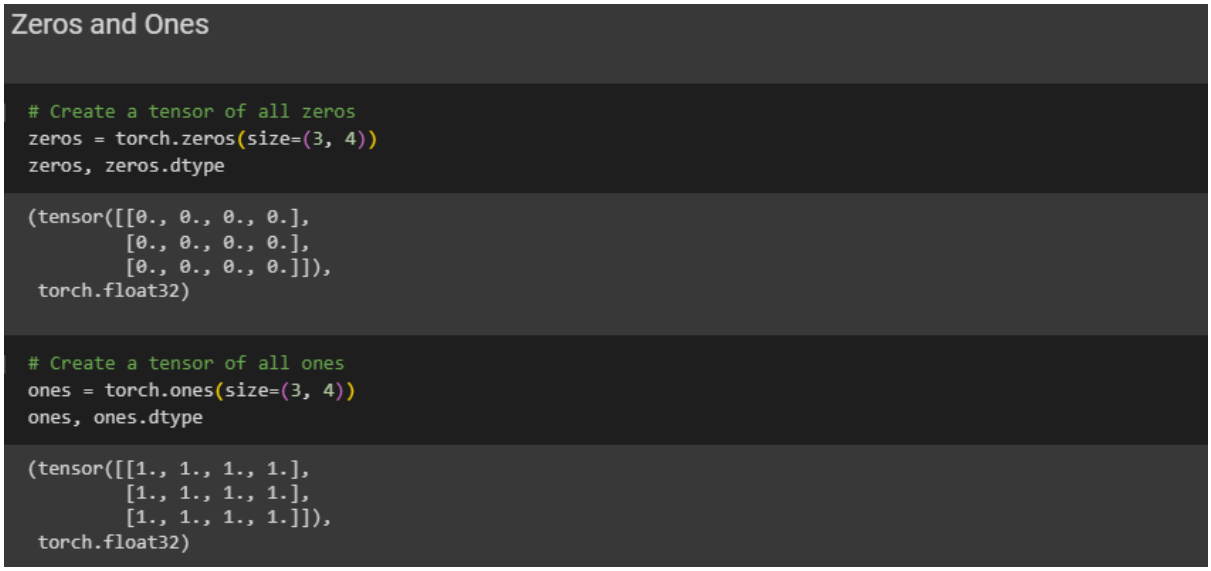
```
# Create a random tensor of size (224, 224, 3)
random_image_size_tensor = torch.rand(size=(224, 224, 3))
random_image_size_tensor.shape, random_image_size_tensor.ndim
```

```
(torch.Size([224, 224, 3]), 3)
```

Kode program melakukan :

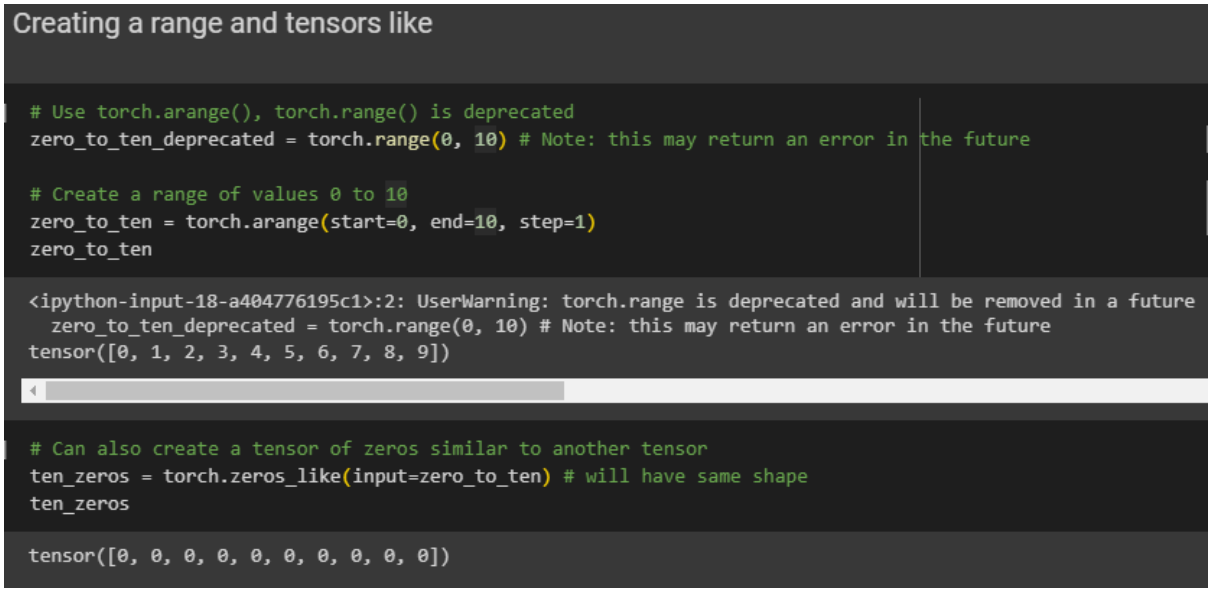
- random_tensor dengan ukuran (3, 4) dibuat menggunakan fungsi torch.rand(). Tensor ini akan berisi nilai-nilai acak antara 0 dan 1. random_tensor, random_tensor.dtype akan menampilkan output random tensor yang telah dibuat.
- random_image_size_tensor dengan ukuran (224, 224, 3) dibuat menggunakan fungsi torch.rand(). Tensor ini akan berisi nilai-nilai acak antara 0 dan 1. random_image_size_tensor.shape, random_image_size_tensor.ndim digunakan untuk

memeriksa bentuk (shape) dan jumlah dimensi dari tensor random_image_size_tensor. Dalam kasus ini, bentuknya adalah (224, 224, 3) dan jumlah dimensinya adalah 3. Ini menunjukkan bahwa tensor tersebut memiliki 224 baris, 224 kolom, dan 3 saluran warna.



Kode program melakukan :

- zeros sebuah tensor dengan ukuran (3, 4) yang berisi semua nilai nol dibuat menggunakan fungsi torch.zeros(). zeros, zeros.dtype akan menampilkan output dari variabel zeros yang berisi semua nilai nol dengan ukuran (3, 4).
- ones sebuah tensor dengan ukuran (3, 4) yang berisi semua nilai satu dibuat menggunakan fungsi torch.ones(). ones, ones.dtype yang akan menampilkna output dari variabel ones yang berisi semua nilai satu dengan ukuran (3, 4).



Kode program melakukan :

- zero_to_ten_deprecated digunakan untuk membuat tensor yang berisi rentang nilai dari 0 hingga 10 menggunakan torch.range(), namun torch.range() telah usang

(deprecated) dan mungkin akan menghasilkan error. Solusinya yaitu menggunakan torch.arange() untuk membuat tensor yang berisi rentang nilai dari 0 hingga 10 yang digunakan untuk menerima fungsi argument start (nilai awal), end (nilai akhir), dan step (selisih antara setiap nilai). Lalu menampilkan output dari variabel zero_to_ten.

- ten_zeros membuat tensor yang berisi semua nilai nol dengan bentuk yang sama seperti tensor input (zero_to_ten) tetapi semua elemennya adalah 0 menggunakan fungsi torch.zeros_like.

Tensor datatypes

```
# Default datatype for tensors is float32
float_32_tensor = torch.tensor([3.0, 6.0, 9.0],
                               dtype=None, # defaults to None, which is torch.float32 or whatever datatype
                               device=None, # defaults to None, which uses the default tensor type
                               requires_grad=False) # if True, operations performed on the tensor are reco

float_32_tensor.shape, float_32_tensor.dtype, float_32_tensor.device
```

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

```
float_16_tensor = torch.tensor([3.0, 6.0, 9.0],
                               dtype=torch.float16) # torch.half would also work

float_16_tensor.dtype
```

```
torch.float16
```

Kode program melakukan :

- float_32_tensor digunakan untuk membuat sebuah tensor dengan nilai [3.0, 6.0, 9.0] dan menggunakan tipe data default (float32) dengan menggunakan fungsi torch.tensor(). float_32_tensor.shape, float_32_tensor.dtype, float_32_tensor.device akan menampilkan output shape, dtype, dan device dari tensor float_32_tensor.
- float_16_tensor digunakan untuk membuat sebuah tensor dengan nilai [3.0, 6.0, 9.0] dan menggunakan tipe data float16 (atau torch.half). float_16_tensor.dtype akan menampilkan output tipe data dari tensor float_16_tensor yaitu float16.

Getting information from tensors

```
# 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}") # will default to CPU
```

```
tensor([[0.3716, 0.8683, 0.5868, 0.9708],
        [0.8098, 0.9688, 0.5461, 0.4897],
        [0.2478, 0.2801, 0.8221, 0.2122]])
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

Kode program membuat sebuah tensor acak dengan ukuran (3, 4) menggunakan fungsi torch.rand() didalam variabel some_tensor. Lalu menampilkan output dari variabel some_tensor, menampilkan bentuk (shape), menampilkan tipe data, dan menampilkan perangkat Dimana tensor some_tensor disimpan.

Basic operations

```
# 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])
```

Kode program melakukan :

- Pada variabel tensor kita menambahkan nilai 10 menggunakan operator penjumlahan. Hasilnya adalah tensor baru dengan nilai [11, 12, 13] dan tensor asli tidak akan berubah dan akan balik ke nilai semula.
- Mengalikan tensor dengan nilai 10 menggunakan operator perkalian. Hasilnya adalah tensor baru dengan nilai [10, 20, 30] dan tensor asli tidak akan berubah dan akan balik ke nilai semula.
- Mengurangi tensor dengan nilai 10 menggunakan operator pengurangan. Hasilnya adalah tensor baru dengan nilai [-9, -8, -7].
- Menambahkan tensor dengan nilai 10 menggunakan operator penjumlahan. Hasilnya adalah tensor baru dengan nilai [1, 2, 3].

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

```
tensor([10, 20, 30])
```

```
# Original tensor is still unchanged
tensor
```

```
tensor([1, 2, 3])
```

```
# Element-wise multiplication (each element multiplies its equivalent, index 0->0, 1->1, 2->2)
print(tensor, "*", tensor)
print("Equals:", tensor * tensor)
```

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

Kode program melakukan :

- Menggunakan fungsi torch.multiply() untuk mengalikan setiap elemen dalam tensor tensor dengan nilai 10. Hasilnya adalah tensor baru dengan nilai [10, 20, 30] dan tensor asli tidak akan berubah dan akan balik ke nilai semula.
- Mencetak operasi perkalian elemen-wise (per-elemen) dari tensor dan mengalikan setiap elemen pada posisi indeks yang sama. Hasilnya adalah [1, 4, 9].

Matrix multiplication

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

```
torch.Size([3])
```

```
# Element-wise matrix multiplication
tensor * tensor
```

```
tensor([1, 4, 9])
```

```
# Matrix multiplication
torch.matmul(tensor, tensor)
```

```
tensor(14)
```

```
# Can also use the "@" symbol for matrix multiplication, though not recommended
tensor @ tensor
```

```
tensor(14)
```

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

```
CPU times: user 1.17 ms, sys: 36 µs, total: 1.21 ms
Wall time: 4.02 ms
tensor(14)
```

Kode program melakukan :

- Menampilkan output dari variabel tensor.shape yaitu bentuk (shape).
- Melakukan perkalian elemen-wise (per-elemen) dari tensor.
- Melakukan perkalian matriks dari tensor tensor menggunakan fungsi torch.matmul().
- Melakukan perkalian matriks dari tensor menggunakan operator @.
- Melakukan perkalian matriks secara manual menggunakan perulangan for.

One of the most common errors in deep learning (shape errors)

```
# 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)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-38-5893e600ebdf> in <cell line: 10>()
      8                         [9, 12]], dtype=torch.float32)
      9
---> 10 torch.matmul(tensor_A, tensor_B) # (this will error)

RuntimeError: mat1 and mat2 shapes cannot be multiplied (3x2 and 3x2)

 SEARCH STACK OVERFLOW
```

Kode program melakukan perkalian matriks antara tensor_A dan tensor_B menggunakan fungsi torch.matmul(). Namun, operasi ini akan menghasilkan kesalahan (error). Kesalahan tersebut terjadi karena bentuk (shape) dari tensor tidak sesuai untuk perkalian matriks yang valid.

```
# 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.]])
```

```
# 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.]])
```

Kode program melakukan :

- Menampilkan output dari variabel tensor_A dan tensor_B secara berurutan.
- Menampilkan output dari tensor_A dan transpose dari tensor_B secara berurutan.

```
# The operation works when tensor_B is transposed
print(f"Original shapes: tensor_A = {tensor_A.shape}, tensor_B = {tensor_B.shape}\n")
print(f"New shapes: tensor_A = {tensor_A.shape} (same as above), tensor_B.T = {tensor_B.T.shape}\n")
print(f"Multiplying: {tensor_A.shape} * {tensor_B.T.shape} <- inner dimensions match\n")
print("Output:\n")
output = torch.matmul(tensor_A, tensor_B.T)
print(output)
print(f"\nOutput shape: {output.shape}")
```

```
Original shapes: tensor_A = torch.Size([3, 2]), tensor_B = torch.Size([3, 2])

New shapes: tensor_A = torch.Size([3, 2]) (same as above), tensor_B.T = torch.Size([2, 3])

Multiplying: torch.Size([3, 2]) * torch.Size([2, 3]) <- inner dimensions match

Output:

tensor([[ 27.,  30.,  33.],
        [ 61.,  68.,  75.],
        [ 95., 106., 117.]])

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

```
# torch.mm is a shortcut for matmul
torch.mm(tensor_A, tensor_B.T)
```

```
tensor([[ 27.,  30.,  33.],
        [ 61.,  68.,  75.],
        [ 95., 106., 117.]])
```

Kode program melakukan :

- Menampilkan output bentuk (shape) dari tensor_A dan tensor_B, bentuk dari transpose dari tensor_B, dan menampilkan langkah-langkah perkalian matriks dan hasil perkaliannya.
- torch.mm digunakan untuk melakukan perkalian matriks antara tensor_A dan transpose dari tensor_B.

```
# Since the linear layer starts with a random weights matrix, let's make it reproducible (more on this later)
torch.manual_seed(42)
# This uses matrix multiplication
linear = torch.nn.Linear(in_features=2, # in_features = matches inner dimension of input
                         out_features=6) # out_features = describes outer value
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, 0.4669, 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])
```

```
# 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()}") # won't work without float datatype
print(f"Sum: {x.sum()}")
```

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

Kode program melakukan :

- Mengatur seed acak menjadi 42 menggunakan fungsi torch.manual_seed(42) yang berfungsi untuk membuat hasil dari operasi yang menggunakan angka acak menjadi

deterministik, sehingga dapat direproduksi. Lalu membuat objek linear yang merupakan layer linear dengan 2 input features dan 6 output features. Lalu mengambil tensor_A sebagai input x dan melakukan operasi forward pass pada layer linear dengan memanggil linear(x).

- Membuat tensor x menggunakan fungsi torch.arange() dengan parameter (start=0, end=100, step=10).
- Menampilkan output nilai minimum, maksimum, rata-rata (mean), dan jumlah (sum) dari tensor x.

```
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
```

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

```
torch.float32
```

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

```
tensor([10., 20., 30., 40., 50., 60., 70., 80., 90.], dtype=torch.float16)
```

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

```
tensor([10, 20, 30, 40, 50, 60, 70, 80, 90], dtype=torch.int8)
```

Kode program melakukan :

- Menghitung nilai maksimum, nilai minimum, rata-rata (mean), dan jumlah (sum) dari tensor x menggunakan fungsi torch.max(), torch.min(), torch.mean(), dan torch.sum().
- Membuat tensor menggunakan fungsi torch.arange() dengan parameter (start=10, end=100, step=10). Outputnya adalah nilai [10, 20, 30, 40, 50, 60, 70, 80, 90]. Lalu menampilkan indeks dimana nilai maksimum dan nilai minimum terjadi pada tensor tensor menggunakan metode .argmax() dan .argmin().
- Membuat tensor dengan tipe data float menggunakan fungsi torch.arange() dengan parameter (start=10., end=100., step=10.)
- Membuat tensor tensor_float16 dengan tipe data float16 menggunakan metode .type(torch.float16).

- membuat tensor tensor_int8 dengan tipe data int8 menggunakan metode .type(torch.int8).

Reshaping, stacking, squeezing and unsqueezing

```
# 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_reshaped = x.reshape(1, 7)
x_reshaped, x_reshaped.shape
```

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

```
# Change view (keeps same data as original but changes view)
# See more: https://stackoverflow.com/a/54507446/7900723
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.]))
```

```
# Stack tensors on top of each other
x_stacked = torch.stack([x, x, x, x], dim=0) # try changing dim to dim=1 and see what happens
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.]])
```

Kode program melakukan :

- Membuat tensor x menggunakan fungsi torch.arange() dengan parameter (start=1., end=8.). Outputnya yaitu nilai [1., 2., 3., 4., 5., 6., 7.] beserta shapenya yaitu 7.
- Mengubah bentuk (shape) dari tensor x menjadi (1, 7) menggunakan metode .reshape(1, 7). Lalu menampilkan output tensor x_reshaped beserta bentuknya.
- Menggunakan metode .view(1, 7) pada tensor x untuk mengubah tampilan (view) tensor menjadi (1, 7) yang disimpan dalam varibel z. Lalu menampilkan output tensor z beserta bentuknya.
- Mengubah nilai elemen pertama dari tensor z menjadi 5. Lalu menampilkan output nilai elemen pertama yang sama.
- Menggabungkan (stack) tensor x sebanyak 4 kali secara bertumpuk (dimensi 0) menjadi tensor x_stacked menggunakan fungsi torch.stack(). Lalu menampilkan output dari tensor x_stacked.

```
print(f"Previous tensor: {x_reshaped}")
print(f"Previous shape: {x_reshaped.shape}")

# Remove extra dimension from x_reshaped
x_squeezed = x_reshaped.squeeze()
print(f"\nNew tensor: {x_squeezed}")
print(f"New shape: {x_squeezed.shape}")
```

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

New tensor: tensor([5., 2., 3., 4., 5., 6., 7.])
New shape: torch.Size([7])
```

```
print(f"Previous tensor: {x_squeezed}")
print(f"Previous shape: {x_squeezed.shape}")

## Add an extra dimension with unsqueeze
x_unsqueezed = x_squeezed.unsqueeze(dim=0)
print(f"\nNew tensor: {x_unsqueezed}")
print(f"New shape: {x_unsqueezed.shape}")
```

```
Previous tensor: tensor([5., 2., 3., 4., 5., 6., 7.])
Previous shape: torch.Size([7])

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

Kode program melakukan :

- Menampilkan output dari tensor x_reshaped sebelumnya dan bentuk (shape), menghapus dimensi tambahan yang ada, dan menampilkan tensor x_squeezed yang baru dan bentuknya.
- Menampilkan output tensor x_squeezed sebelumnya dan bentuk (shape), menggunakan metode .unsqueeze(dim=0) pada tensor x_squeezed untuk menambahkan dimensi tambahan dengan ukuran 1, dan menampilkan tensor x_unsqueezed yang baru dan bentuknya.

```
# Create tensor with specific shape
x_original = torch.rand(size=(224, 224, 3))

# Permute the original tensor to rearrange the axis order
x_permuted = x_original.permute(2, 0, 1) # shifts axis 0->1, 1->2, 2->0

print(f"Previous shape: {x_original.shape}")
print(f"New shape: {x_permuted.shape}")
```

```
Previous shape: torch.Size([224, 224, 3])
New shape: torch.Size([3, 224, 224])
```

Kode program membuat tensor x_original dengan bentuk (shape) (224, 224, 3) menggunakan fungsi torch.rand(). Lalu menggunakan metode .permute() pada tensor x_original dengan parameter (2, 0, 1) untuk mengubah urutan sumbu (axis). Lalu menampilkan output bentuk (shape) sebelumnya dari tensor x_original dan bentuk (shape) baru dari tensor x_permuted.

Indexing (selecting data from tensors)

```
# 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]))
```

```
# Let's index bracket by bracket
print(f"First square bracket:\n{x[0]}")
print(f"Second square bracket: {x[0][0]}")
print(f"Third square bracket: {x[0][0][0]}")
```

```
First square bracket:
tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
Second square bracket: tensor([1, 2, 3])
Third square bracket: 1
```

```
# Get all values of 0th dimension and the 0 index of 1st dimension
x[:, 0]
```

```
tensor([[1, 2, 3]])
```

```
# Get all values of 0th & 1st dimensions but only index 1 of 2nd dimension
x[:, :, 1]
```

```
tensor([[2, 5, 8]])
```

Kode program melakukan :

- Membuat tensor x menggunakan fungsi torch.arange() dengan parameter (start=1, end=10) dan kemudian mengubah bentuk (shape) menjadi (1, 3, 3) menggunakan .reshape(). Lalu menampilkan output tensor x beserta bentuk (shape)-nya.
- Menampilkan output elemen tensor x dengan menggunakan indeks kurung siku (bracket indexing). x[0] mencetak elemen-elemen pada indeks pertama dari dimensi pertama, x[0][0] mencetak elemen-elemen pada indeks pertama dari dimensi pertama dan kedua, dan x[0][0][0] mencetak elemen pertama dari tensor.
- Mengambil semua nilai pada dimensi pertama (indeks 0) dan indeks 0 pada dimensi kedua dari tensor x.
- Mengambil semua nilai pada dimensi pertama dan kedua, tetapi hanya indeks 1 pada dimensi ketiga dari tensor x.

```
# Get all values of the 0 dimension but only the 1 index value of the 1st and 2nd dimension
x[:, 1, 1]
```

```
tensor([5])
```

```
# Get index 0 of 0th and 1st dimension and all values of 2nd dimension
x[0, 0, :] # same as x[0][0]
```

```
tensor([1, 2, 3])
```

Kode program melakukan :

- Mengambil semua nilai pada dimensi pertama (indeks 0), tetapi hanya nilai dengan indeks 1 pada dimensi kedua dan dimensi ketiga dari tensor x.
- Mengambil nilai dengan indeks 0 pada dimensi pertama dan kedua, tetapi semua nilai pada dimensi ketiga dari tensor x. Ini sama dengan x[0][0].

PyTorch tensors & NumPy

```
# NumPy array to tensor
import torch
import numpy as np
array = np.arange(1.0, 8.0)
tensor = torch.from_numpy(array)
array, tensor
```

```
(array([1., 2., 3., 4., 5., 6., 7.]),
 tensor([1., 2., 3., 4., 5., 6., 7.], dtype=torch.float64))
```

```
# Change the array, keep the tensor
array = array + 1
array, tensor
```

```
(array([2., 3., 4., 5., 6., 7., 8.]),
 tensor([1., 2., 3., 4., 5., 6., 7.], dtype=torch.float64))
```

Kode program melakukan :

- Membuat variabel array menggunakan fungsi np.arange() dengan parameter (start=1.0, end=8.0). Lalu mengonversi array NumPy tersebut menjadi tensor PyTorch menggunakan torch.from_numpy(). Lalu menampilkan output array dan tensor.
- Mengubah nilai-nilai dalam array NumPy dengan menambahkan 1 ke setiap elemennya. Lalu menampilkan array dan tensor.

```
# Tensor to NumPy array
tensor = torch.ones(7) # create a tensor of ones with dtype=float32
numpy_tensor = tensor.numpy() # will be dtype=float32 unless changed
tensor, numpy_tensor
```

```
(tensor([1., 1., 1., 1., 1., 1., 1.]),
 array([1., 1., 1., 1., 1., 1., 1.], dtype=float32))
```

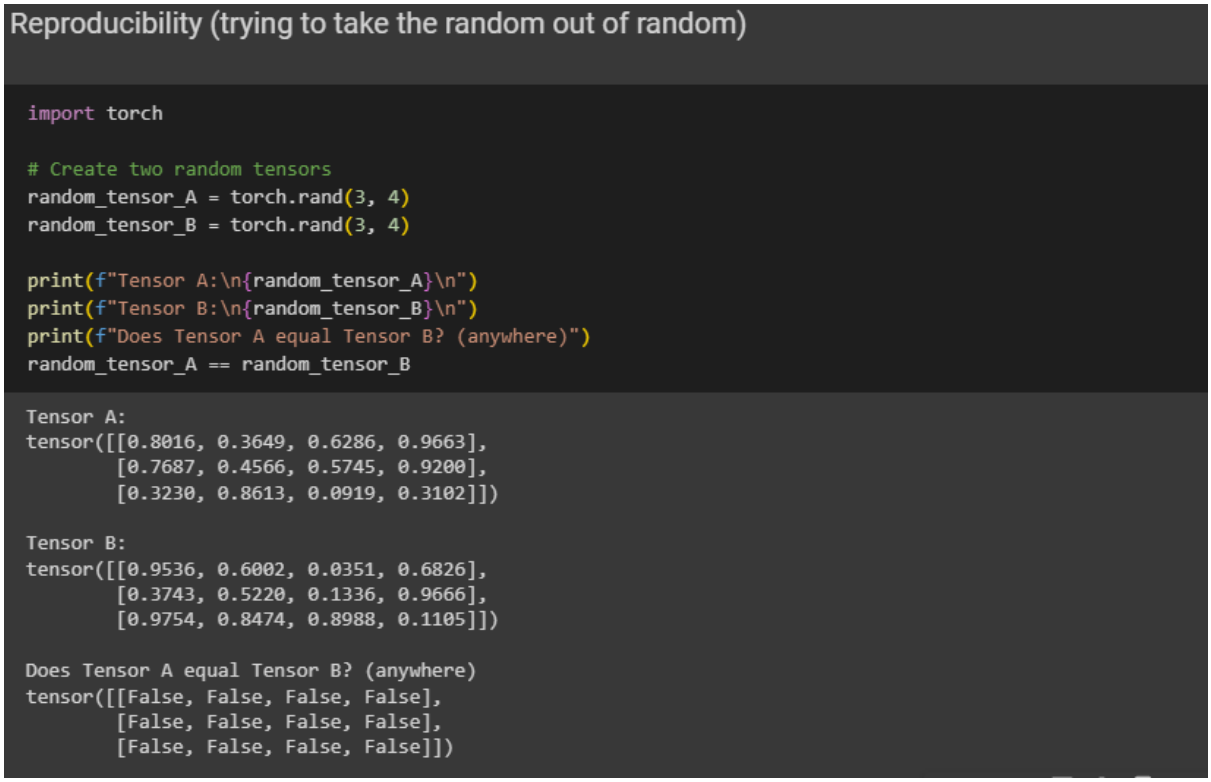
```
# Change the tensor, keep the array the same
tensor = tensor + 1
tensor, numpy_tensor
```

```
(tensor([2., 2., 2., 2., 2., 2., 2.]),
 array([1., 1., 1., 1., 1., 1., 1.], dtype=float32))
```

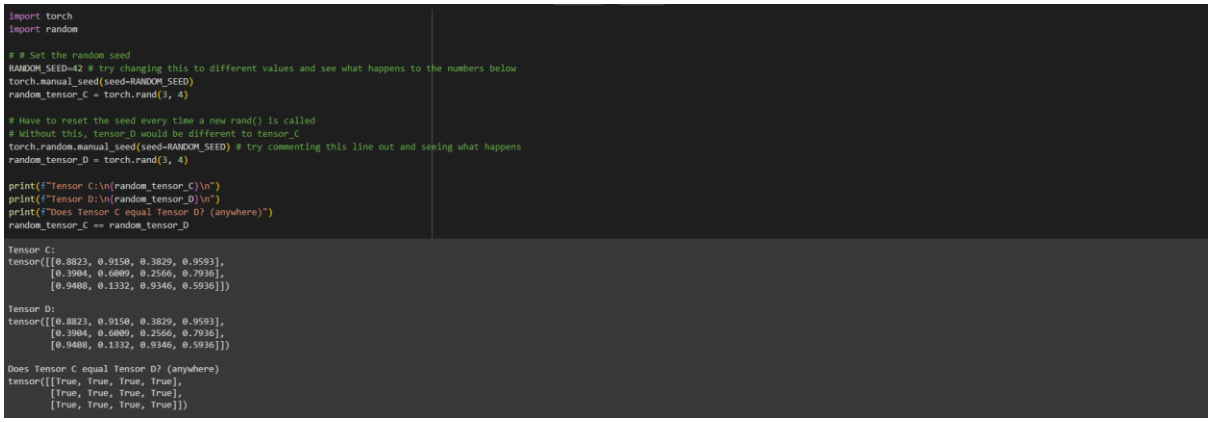
Kode program melakukan :

- Membuat tensor PyTorch tensor yang berisi nilai 1 menggunakan fungsi torch.ones() dengan parameter (7). Lalu mengonversi tensor PyTorch tersebut menjadi array NumPy menggunakan metode .numpy(). Lalu menampilkan output tensor dan numpy_tensor.

- Mengubah nilai-nilai dalam tensor PyTorch dengan menambahkan 1 ke setiap elemennya. Lalu menampilkan tensor dan numpy_tensor.



Kode program membuat dua tensor acak random_tensor_A dan random_tensor_B menggunakan fungsi torch.rand() dengan bentuk (shape) 3x4. Lalu menampilkan output kedua tensor tersebut dan menampilkan output apakah Tensor A sama dengan Tensor B (di mana pun dalam tensor).



Kode program melakukan menggunakan torch.manual_seed() dengan nilai RANDOM_SEED yang ditentukan yaitu 42. Lalu membuat tensor acak random_tensor_C menggunakan torch.rand() dengan bentuk (shape) 3x4. Lalu untuk random_tensor_D menggunakan torch.random.manual_seed() sebelum membuat tensor acak kedua. Lalu menampilkan kedua tensor tersebut dan menampilkan apakah Tensor C sama dengan Tensor D (di mana pun dalam tensor).

Running tensors on GPUs

+ Code + Markdown

Click here to ask Blackbox to help you code faster |

```
!nvidia-smi
```

✓ 1.0s

```
Fri Jan  5 04:49:32 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 537.13                 Driver Version: 537.13       CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                     TCC/WDDM  | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce GTX 1050 Ti   WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   48C    P8              N/A / ERR! |      0MiB /  4096MiB |      1%      Default |
|                                         |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2920    C+G   ...inaries\Win64\EpicGamesLauncher.exe    N/A      |
+-----------------------------------------------------------------------------+
```

Kode program memeriksa informasi GPU pada sistem yang menggunakan perangkat GPU Nvidia.

Click here to ask Blackbox to help you code faster |

```
# Check for GPU
import torch
torch.cuda.is_available()
```

✓ 0.0s

```
False
```

Click here to ask Blackbox to help you code faster |

```
# Set device type
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

✓ 0.0s

```
'cpu'
```

Click here to ask Blackbox to help you code faster |

```
# Count number of devices
torch.cuda.device_count()
```

✓ 0.0s

```
0
```

Kode program melakukan :

- Memeriksa apakah GPU tersedia dalam sistem menggunakan fungsi torch.cuda.is_available().
- Mengatur jenis perangkat (device) yang akan digunakan untuk komputasi menggunakan GPU jika GPU tersedia, dan menggunakan CPU jika tidak tersedia.
- menghitung jumlah perangkat GPU yang tersedia dalam sistem menggunakan torch.cuda.device_count().

```
Click here to ask Blackbox to help you code faster |
# Create tensor (default on CPU)
tensor = torch.tensor([1, 2, 3])

# Tensor not on GPU
print(tensor, tensor.device)

# Move tensor to GPU (if available)
tensor_on_gpu = tensor.to(device)
tensor_on_gpu
```

✓ 0.0s

```
tensor([1, 2, 3]) cpu

tensor([1, 2, 3])
```

```
Click here to ask Blackbox to help you code faster |
# If tensor is on GPU, can't transform it to NumPy (this will error)
tensor_on_gpu.numpy()
```

✓ 0.0s

```
array([1, 2, 3], dtype=int64)
```

```
Click here to ask Blackbox to help you code faster |
# Instead, copy the tensor back to cpu
tensor_back_on_cpu = tensor_on_gpu.cpu().numpy()
tensor_back_on_cpu
```

✓ 0.0s

```
array([1, 2, 3], dtype=int64)
```

Kode program melakukan :

- Membuat tensor menggunakan torch.tensor() dengan nilai [1, 2, 3]. Lalu menampilkan output tensor dan tensor.device. Lalu memindahkan tensor ke GPU (jika tersedia) menggunakan metode .to() dengan argumen device.
- Mengonversi tensor yang berada di GPU menjadi array NumPy.
- Menyalin tensor kembali ke CPU menggunakan metode .cpu().

```
Click here to ask Blackbox to help you code faster |
tensor_on_gpu
```

✓ 0.0s

```
tensor([1, 2, 3])
```

Kode program melakukan pemindahan tensor ke GPU (jika tersedia). Jika tidak, akan tetap berada di CPU.

PyTorch Workflow

PyTorch Workflow adalah serangkaian langkah yang biasanya diikuti saat menggunakan framework PyTorch untuk membangun dan melatih model pembelajaran mesin. Alur kerja ini mencakup beberapa tahap yang meliputi persiapan data, membangun model, melatih model, dan melakukan evaluasi.

Pada PyTorch Workflow ini akan membahas tentang :

- Getting data ready: Data dapat berupa apa saja, tetapi untuk memulai, kita akan membuat garis lurus sederhana.
- Building a model: Di sini, kita akan membuat model untuk mempelajari pola dalam data. Kita juga akan memilih fungsi kerugian (loss function), pengoptimasi (optimizer), dan membangun loop pelatihan (training loop).
- Fitting the model to data (training): Kita sudah memiliki data dan model, sekarang mari kita biarkan model (mencoba) menemukan pola dalam data pelatihan.

- Making predictions and evaluating a model (inference): Model kita telah menemukan pola dalam data, mari kita bandingkan temuannya dengan data aktual (pengujian).
- Saving and loading a model: Anda mungkin ingin menggunakan model Anda di tempat lain atau kembali ke model tersebut nanti, di sini kita akan membahas cara melakukannya.
- Putting it all together: Mari kita gabungkan semua langkah di atas.

1. Data (preparing and loading)

```
# 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],
         [0.0200],
         [0.0400],
         [0.0600],
         [0.0800],
         [0.1000],
         [0.1200],
         [0.1400],
         [0.1600],
         [0.1800]]),
 tensor([[0.3000],
         [0.3140],
         [0.3280],
         [0.3420],
         [0.3560],
         [0.3700],
         [0.3840],
         [0.3980],
         [0.4120],
         [0.4260]]))
```

Kode program membuat data berupa garis lurus sederhana dengan parameter weight (0.7) yang menentukan kemiringan garis lurus, dan bias (0.3) yang menentukan pergeseran garis lurus. Menggunakan parameter-parameter tersebut untuk menghasilkan data X dan y. Data X adalah deret nilai dari 0 hingga 1 dengan interval 0.02, sedangkan data y dihasilkan dengan mengalikan setiap nilai X dengan weight dan menambahkan bias.

```
# Create train/test split
train_split = int(0.8 * len(X)) # 80% of data used for training set, 20% for testing
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)
```

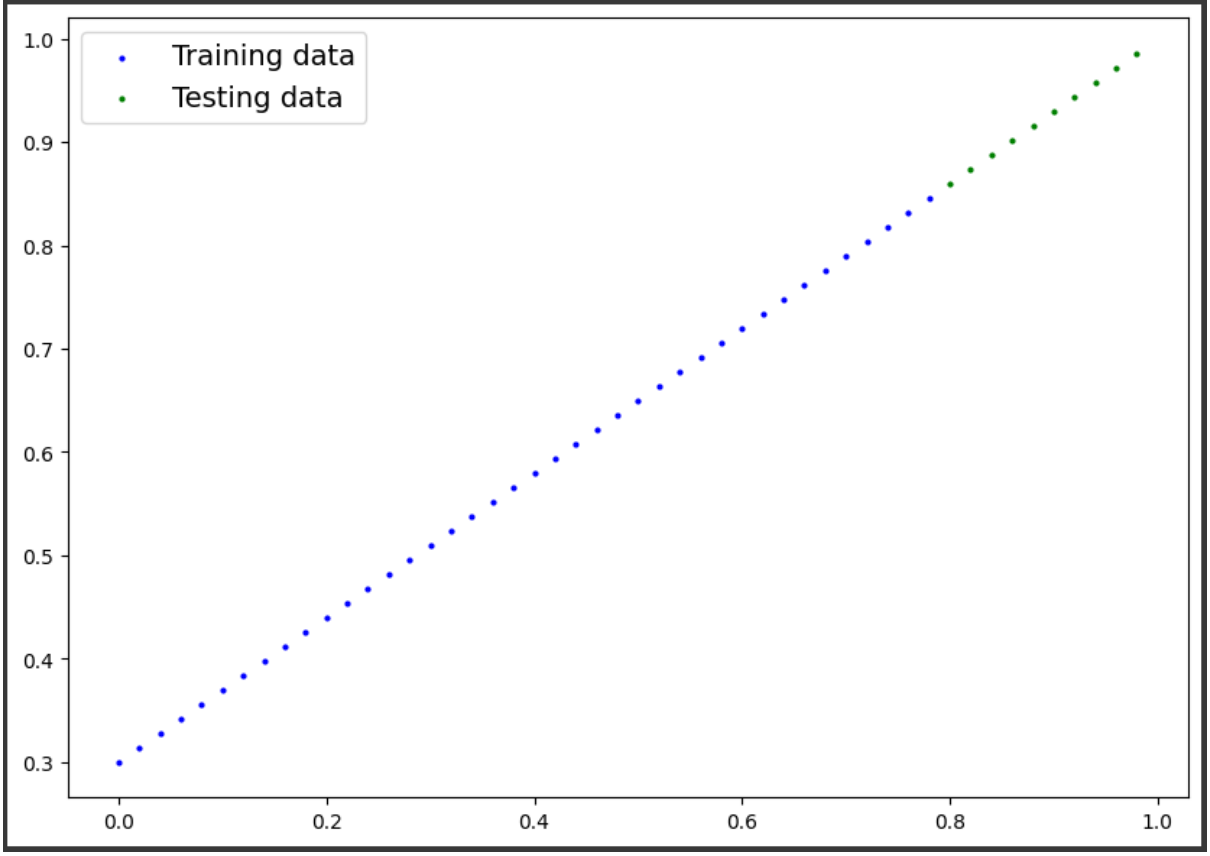
Kode program melakukan perhitungan jumlah indeks data yang akan digunakan untuk data latih, m emisahkan sebagian pertama data X dan y sesuai dengan jumlah indeks yang

ditentukan oleh train_split, memisahkan sisa data X dan y setelah data latih, dan menghitung panjang (jumlah elemen) dari masing-masing data latih dan data uji.

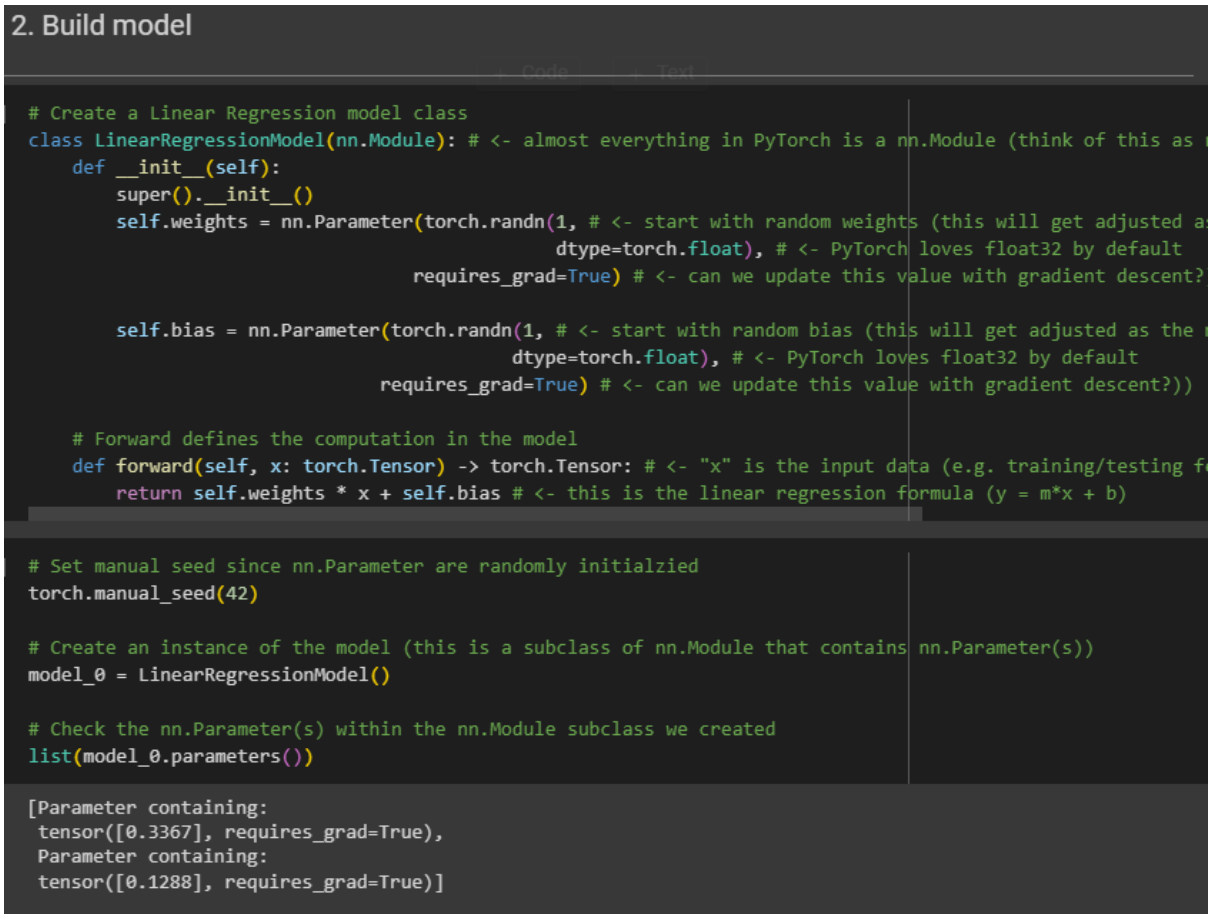
```
def plot_predictions(train_data=X_train,
                     train_labels=y_train,
                     test_data=X_test,
                     test_labels=y_test,
                     predictions=None):
  """
  Plots training data, test data and compares predictions.
  """
  plt.figure(figsize=(10, 7))

  # Plot training data in blue
  plt.scatter(train_data, train_labels, c="b", s=4, label="Training data")

  # Plot test data in green
  plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")

  if predictions is not None:
    # Plot the predictions in red (predictions were made on the test data)
    plt.scatter(test_data, predictions, c="r", s=4, label="Predictions")

  # Show the legend
  plt.legend(prop={"size": 14});
```

Kode program membuat plot untuk menghasilkan sebuah grafik yang membandingkan data latih, data uji, dan prediksi (jika ada).

```
plot_predictions();
```

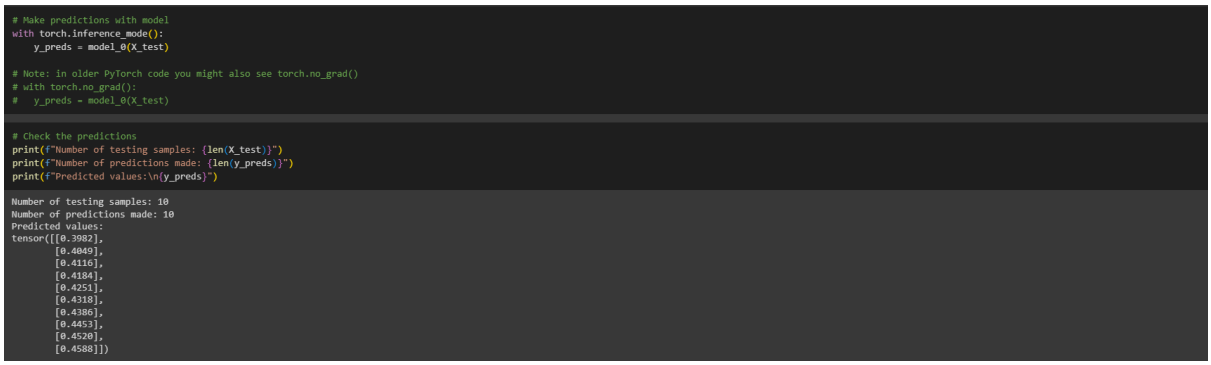


Kode program melakukan pemanggilan fungsi plot_predictions() dan menampilkan hasil plot dari training data dan testing data.



```
2. Build model

# Create a Linear Regression model class
class LinearRegressionModel(nn.Module): # <- almost everything in PyTorch is a nn.Module (think of this as
    def __init__(self):
        super().__init__()
        self.weights = nn.Parameter(torch.randn(1, # <- start with random weights (this will get adjusted a
                                                dtype=torch.float), # <- PyTorch loves float32 by default
                                    requires_grad=True) # <- can we update this value with gradient descent?

        self.bias = nn.Parameter(torch.randn(1, # <- start with random bias (this will get adjusted as the
                                             dtype=torch.float), # <- PyTorch loves float32 by default
                                 requires_grad=True) # <- can we update this value with gradient descent?))

    # Forward defines the computation in the model
    def forward(self, x: torch.Tensor) -> torch.Tensor: # <- "x" is the input data (e.g. training/testing f
        return self.weights * x + self.bias # <- this is the linear regression formula (y = m*x + b)

# Set manual seed since nn.Parameter are randomly initialzied
torch.manual_seed(42)

# Create an instance of the model (this is a subclass of nn.Module that contains nn.Parameter(s))
model_0 = LinearRegressionModel()

# Check the nn.Parameter(s) within the nn.Module subclass we created
list(model_0.parameters())

[Parameter containing:
 tensor([0.3367], requires_grad=True),
 Parameter containing:
 tensor([0.1288], requires_grad=True)]
```

Kode program mengatur seed acak agar hasil inisialisasi parameter menjadi deterministik, sehingga dapat direproduksi dan mengembalikan daftar parameter yang ada dalam model. Dalam hal ini, daftarnya akan berisi parameter weights dan bias, yang keduanya merupakan objek nn.Parameter.

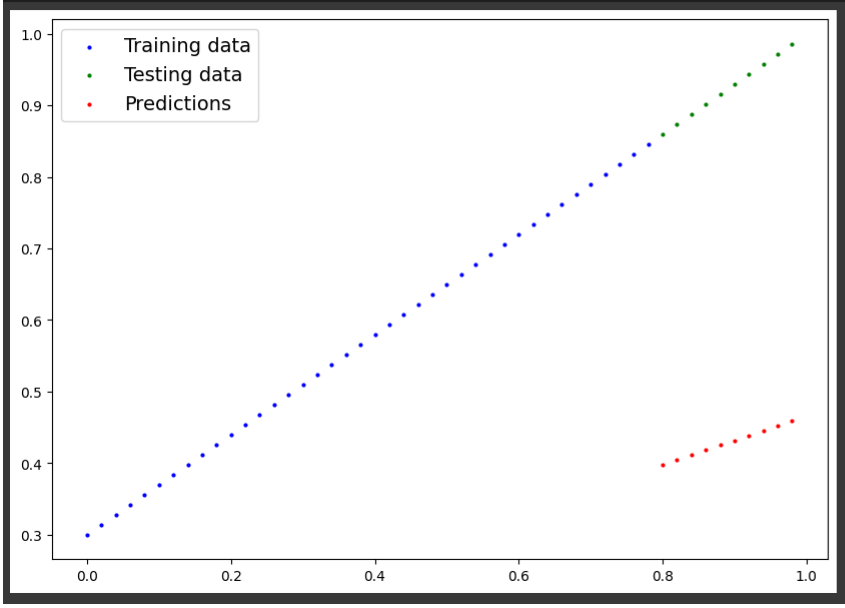


```
# Make predictions with model
with torch.inference_mode():
    y_preds = model_0(X_test)

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

# Check the predictions
print(f"Number of testing samples: {len(X_test)}")
print(f"Number of predictions made: {len(y_preds)}")
print(f"Predicted values:\n{y_preds}")

Number of testing samples: 10
Number of predictions made: 10
Predicted values:
tensor([[0.3982],
        [0.4049],
        [0.4116],
        [0.4184],
        [0.4251],
        [0.4318],
        [0.4386],
        [0.4453],
        [0.4520],
        [0.4588]])
```

Kode program melakukan :

- Mengaktifkan mode inferensi pada PyTorch dengan fungsi torch.inference_mode(). Variabel y_preds digunakan untuk menyimpan hasil prediksi dari model model_0 terhadap data X_test.
- Menampilkan jumlah sampel data uji, jumlah prediksi yang dibuat oleh model, dan nilai-nilai yang diprediksi oleh model untuk setiap sampel data uji.

```
plot_predictions(predictions=y_preds)
```



Kode program melakukan pemanggilan plot_predictions(predictions=y_preds).

```
y_test - y_preds
```

```
tensor([[0.4618],
        [0.4691],
        [0.4764],
        [0.4836],
        [0.4909],
        [0.4982],
        [0.5054],
        [0.5127],
        [0.5200],
        [0.5272]])
```

Kode program menampilkan hasil selisih antara y_test dan y_preds dalam bentuk tensor.

3. Train model

```
# Create the loss function
loss_fn = nn.L1Loss() # MAE loss is same as L1Loss

# Create the optimizer
optimizer = torch.optim.SGD(params=model_0.parameters(), # parameters of target model to optimize
                            lr=0.01) # learning rate (how much the optimizer should change parameters at ea
```

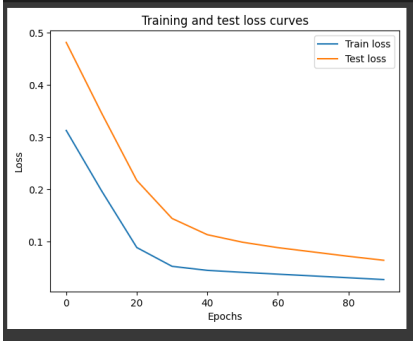
Kode program membuat fungsi loss dengan menggunakan Mean Absolute Error (MAE) loss yang digunakan untuk mengukur selisih absolut antara prediksi dan target. Lalu membuat optimizer dengan menggunakan metode Stochastic Gradient Descent (SGD yaitu metode optimisasi yang umum digunakan dalam pelatihan model.

```python
torch.manual_seed(42)

# Set the number of epochs (how many times the model will pass over the training data)
epochs = 100

# Create empty loss lists to track values
train_loss_values = []
test_loss_values = []
epoch_count = []

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 datatype, so comparisons need to be done with tensors of the same type

      # 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} ")
```

```
Epoch: 0 | MAE Train Loss: 0.31288138031959534 | MAE Test Loss: 0.48106518387794495
Epoch: 10 | MAE Train Loss: 0.1976713240146637 | MAE Test Loss: 0.3463551998138428
Epoch: 20 | MAE Train Loss: 0.08908725529909134 | MAE Test Loss: 0.21729660034179688
Epoch: 30 | MAE Train Loss: 0.053148526698350906 | MAE Test Loss: 0.14464017748832703
Epoch: 40 | MAE Train Loss: 0.04543796554207802 | MAE Test Loss: 0.11360953003168106
Epoch: 50 | MAE Train Loss: 0.04167863354086876 | MAE Test Loss: 0.09919948130846024
Epoch: 60 | MAE Train Loss: 0.03818932920694351 | MAE Test Loss: 0.08886633068323135
Epoch: 70 | MAE Train Loss: 0.03476089984178543 | MAE Test Loss: 0.0805937647819519
Epoch: 80 | MAE Train Loss: 0.03132382780313492 | MAE Test Loss: 0.07232122868299484
Epoch: 90 | MAE Train Loss: 0.02788739837706089 | MAE Test Loss: 0.06473556160926819
```

Kode program melatih model model_0 selama 100 epoch dan menampilkan output pada setiap epoch ke-10.

```python
# Plot the loss curves
plt.plot(epoch_count, train_loss_values, label="Train loss")
plt.plot(epoch_count, test_loss_values, label="Test loss")
plt.title("Training and test loss curves")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend();
```



Kode program membuat plot untuk pelatihan dan pengujian loss dan memvisualisasikan bagaimana loss pada data pelatihan dan pengujian berubah seiring dengan jumlah epoch yang dilakukan.

```python
# Find our model's learned parameters
print("The model learned the following values for weights and bias:")
print(model_0.state_dict())
print("\nAnd the original values for weights and bias are:")
print(f"weights: {weight}, bias: {bias}")
```

```
The model learned the following values for weights and bias:
OrderedDict([('weights', tensor([0.5784])), ('bias', tensor([0.3513]))])

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

Kode program menampilkan nilai-nilai parameter yang telah dipelajari oleh model serta nilai-nilai asli untuk bobot dan bias sebelum model dilatih.

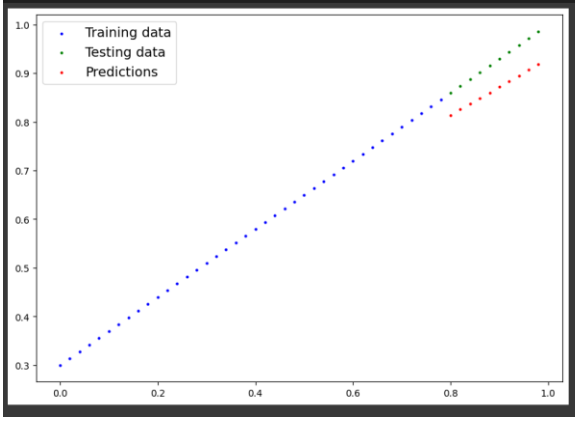
4. Making predictions with a trained PyTorch model (inference)

```
# 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 and data on the same device
  # in our case, we haven't setup device-agnostic code yet so our data and model are
  # 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]])
```

Kode program melakukan inferensi dengan model yang telah dilatih pada data X_test dan mendapatkan hasil prediksi dalam variabel y_preds.

```
plot_predictions(predictions=y_preds)
```



Kode program melakukan pemanggilan plot_predictions(predictions=y_preds) setelah dilakukan pelatihan model.

5. Saving and loading a PyTorch model

```
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_workflow_model_0.pth"
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_0.state_dict(), # only saving the state_dict() only saves the models learned parameters
           f=MODEL_SAVE_PATH)
```

```
Saving model to: models/01_pytorch_workflow_model_0.pth
```

```
# Check the saved file path
!ls -l models/01_pytorch_workflow_model_0.pth
```

```
-rw-r--r-- 1 root root 1680 Jan  5 02:07 models/01_pytorch_workflow_model_0.pth
```

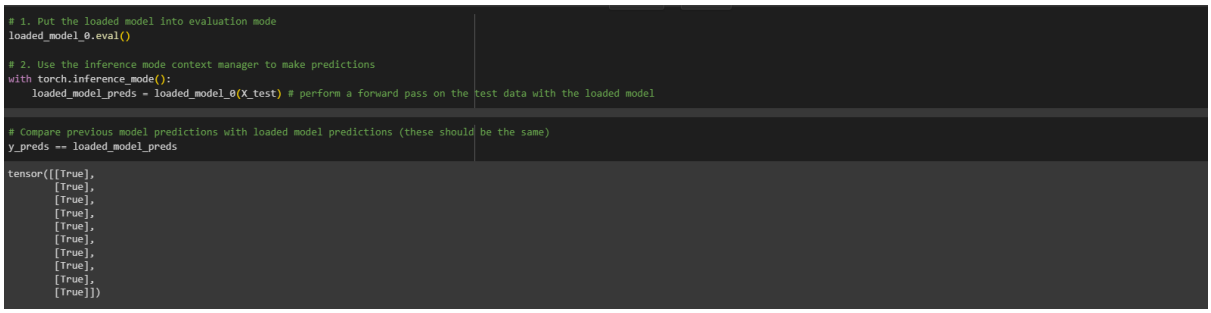
```
# Instantiate a new instance of our model (this will be instantiated with random weights)
loaded_model_0 = LinearRegressionModel()

# Load the state_dict of our saved model (this will update the new instance of our model with trained weights)
loaded_model_0.load_state_dict(torch.load(f=MODEL_SAVE_PATH))
```

```
<All keys matched successfully>
```

Kode program melakukan :

- Membuat direktori "models" jika belum ada. Parameter parents=True akan membuat direktori secara rekursif jika diperlukan. Lalu menyimpan nama file untuk model yang akan disimpan. Dalam contoh, nama file adalah "01_pytorch_workflow_model_0.pth". Lalu membuat Path yang merepresentasikan path lengkap untuk menyimpan model dan torch.save menyimpan state_dict dari model (model_0) ke file yang ditentukan oleh MODEL_SAVE_PATH.

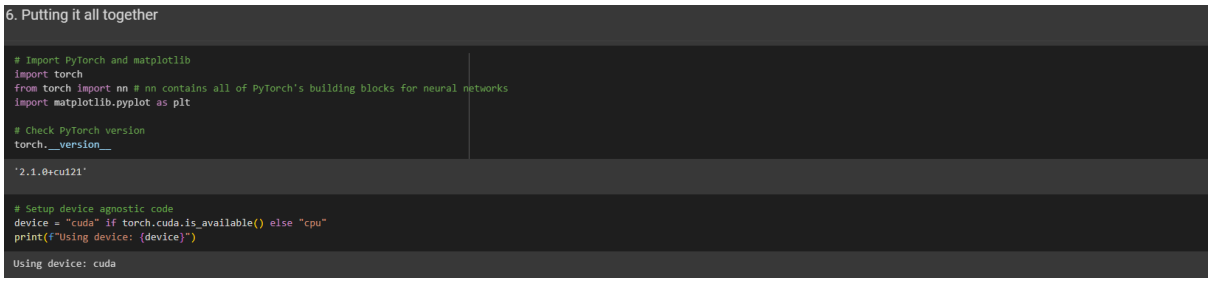
- Menampilkan output path dengan model yang disimpan.
- Menampilkan informasi tentang file yang telah disimpan menggunakan perintah ls -l.
- Membuat instance baru dari model LinearRegressionModel() yang belum dilatih yang digunakan untuk memuat parameter-parameter yang telah dilatih.
- Memuat state_dict dari model yang telah disimpan sebelumnya (MODEL_SAVE_PATH) ke instance loaded_model_0.



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

# 2. Use the inference mode context manager to make predictions
with torch.inference_mode():
    loaded_model_preds = loaded_model_0(X_test) # perform a forward pass on the test data with the loaded model

# Compare previous model predictions with loaded model predictions (these should be the same)
y_preds == loaded_model_preds
```

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

Kode program membandingkan hasil prediksi dari model sebelumnya dengan hasil prediksi dari model yang telah dimuat kembali. Jika kedua hasil prediksi tersebut sama akan ini menunjukkan bahwa model yang telah dilatih sebelumnya telah berhasil dimuat kembali dengan benar.



6. Putting it all together

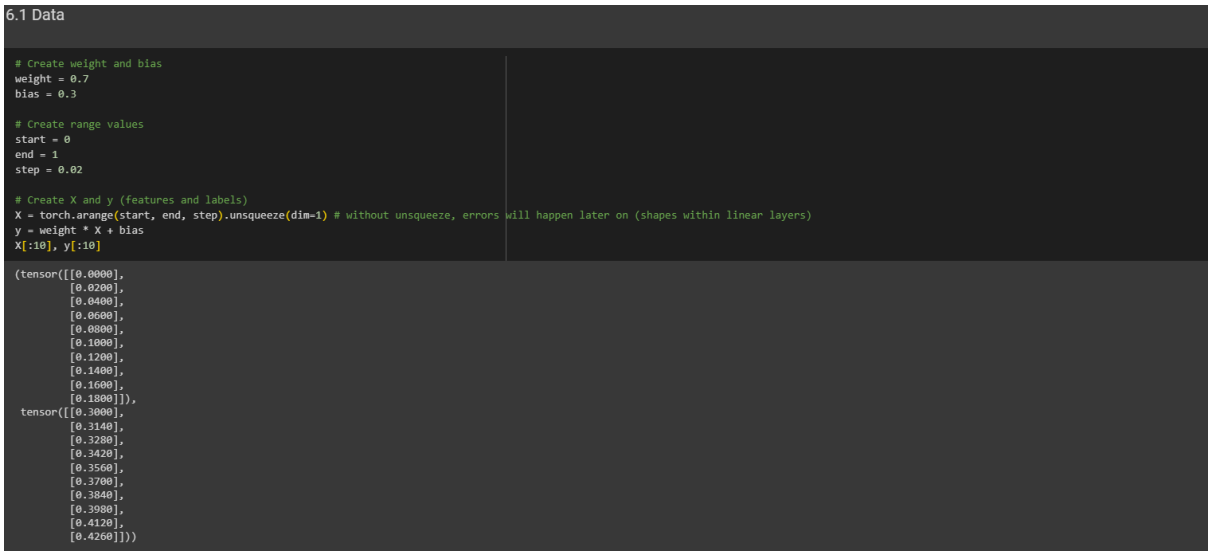
```
# Import PyTorch and matplotlib
import torch
from torch import nn # nn contains all of PyTorch's building blocks for neural networks
import matplotlib.pyplot as plt

# Check PyTorch version
torch.__version__
```

```
'2.1.0+cu121'
```

```
# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using device: {device}")
```

```
Using device: cuda
```

Kode program menampilkan versi PyTorch dan menampilkan perangkat yang akan digunakan dari variabel device.



6.1 Data

```
# 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) # without unsqueeze, errors will happen later on (shapes within linear layers)
y = weight * X + bias
X[:10], y[:10]
```

```
(tensor([[0.0000],
         [0.0200],
         [0.0400],
         [0.0600],
         [0.0800],
         [0.1000],
         [0.1200],
         [0.1400],
         [0.1600],
         [0.1800]]),
 tensor([[0.3000],
         [0.3140],
         [0.3280],
         [0.3420],
         [0.3560],
         [0.3700],
         [0.3840],
         [0.3980],
         [0.4120],
         [0.4260]]))
```

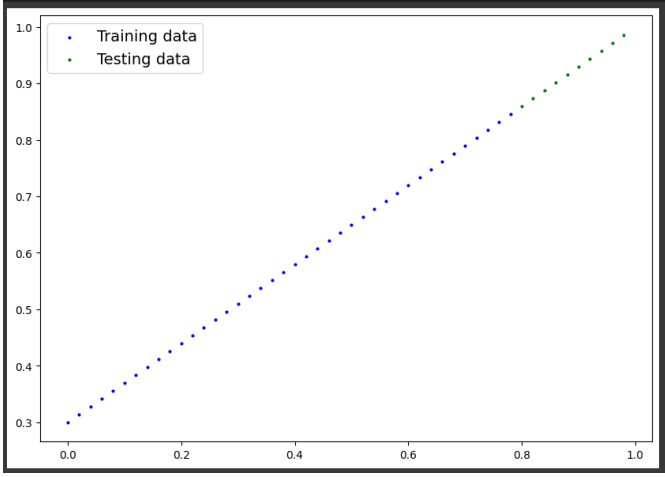
Kode program membuat bobot (weight) dan bias, serta membuat nilai rentang (range values) untuk digunakan dalam pembuatan fitur (X) dan label (y).

```
# 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)
```

Kode program membagi data menjadi data latih (train) dan data uji (test) berdasarkan proporsi yang ditentukan (80% data latih dan 20% data uji) yang berguna untuk melakukan evaluasi dan validasi model pada data yang tidak digunakan selama pelatihan.

```
# Note: If you've reset your runtime, this function won't work,
# you'll have to rerun the cell above where it's instantiated.
plot_predictions(X_train, y_train, X_test, y_test)
```



Kode program melakukan pemanggilan plot_predictions(X_train, y_train, X_test, y_test) setelah dilakukan pelatihan model.

6.2 Building a PyTorch linear model

```
# Subclass nn.Module to make our model
class LinearRegressionModelV2(nn.Module):
    def __init__(self):
        super().__init__()
        # Use nn.Linear() for creating the model parameters
        self.linear_layer = nn.Linear(in_features=1,
                                      out_features=1)

    # Define the forward computation (input data x flows through nn.Linear())
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.linear_layer(x)

# Set the manual seed when creating the model (this isn't always need but is used for demonstrative purposes, try commenting it out and seeing what happens)
torch.manual_seed(42)
model_1 = LinearRegressionModelV2()
model_1, model_1.state_dict()
```

```
(LinearRegressionModelV2(
   (linear_layer): Linear(in_features=1, out_features=1, bias=True)
 ),
 OrderedDict([('linear_layer.weight', tensor([[0.7645]])),
              ('linear_layer.bias', tensor([0.8300])])))
```

```
# Check model device
next(model_1.parameters()).device
```

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

```
# Set model to GPU if it's availalble, otherwise it'll default to CPU
model_1.to(device) # the device variable was set above to be "cuda" if available or "cpu" if not
next(model_1.parameters()).device
```

```
device(type='cuda', index=0)
```

Kode program mengimplementasikan model regresi linear. Subclass LinearRegressionModelV2 dibuat dengan menggunakan nn.Module dan memiliki layer linear untuk melakukan komputasi dan model akan dijalankan pada GPU jika tersedia.

6.3 Training

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

# Create optimizer
optimizer = torch.optim.SGD(params=model_1.parameters(), # optimize newly created model's parameters
                            lr=0.01)
```

Kode program melakukan pengujian dengan menggunakan data pengujian (test_inputs). Kita menggunakan model dalam mode evaluasi (model_1.eval()) dan menghitung loss pada data pengujian (test_outputs dan test_targets).

```
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}")
```

```
Epoch: 0 | Train loss: 0.5551779866218567 | Test loss: 0.5739762187004089
Epoch: 100 | Train loss: 0.006215683650225401 | Test loss: 0.014086711220443249
Epoch: 200 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 300 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 400 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 500 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 600 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 700 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 800 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
Epoch: 900 | Train loss: 0.0012645035749301314 | Test loss: 0.013801801018416882
```

Kode program melakukan pelatihan model menggunakan data pelatihan dan evaluasi model menggunakan data pengujian. Proses pelatihan bertujuan untuk mengoptimalkan parameter-parameter model agar dapat memberikan prediksi yang sesuai dengan target yang diinginkan dengan jumlah epoch 1000.

```
# Find our model's learned parameters
from pprint import pprint # pprint = pretty print, see: https://docs.python.org/3/library/pprint.html
print("The model learned the following values for weights and bias:")
pprint(model_1.state_dict())
print("\nAnd the original values for weights and bias are:")
print(f"weights: {weight}, bias: {bias}")
```

```
The model learned the following values for weights and bias:
OrderedDict([('linear_layer.weight', tensor([[0.6968]], device='cuda:0')),
             ('linear_layer.bias', tensor([0.3025], device='cuda:0'))])

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

Kode program melakukan:

- Mengimpor fungsi pprint dari modul pprint. Pprint yang merupakan singkatan dari "pretty print" dan digunakan untuk memformat output agar lebih mudah dibaca dan terorganisir.
- Menampilkan output yang menunjukkan bahwa nilai-nilai berikut adalah parameter-parameter yang dipelajari oleh model menggunakan pprint untuk mencetak state dictionary dari model (model_1.state_dict()).
- Menampilkan output yang menunjukkan bahwa nilai-nilai berikut adalah nilai asli dari bobot dan bias.

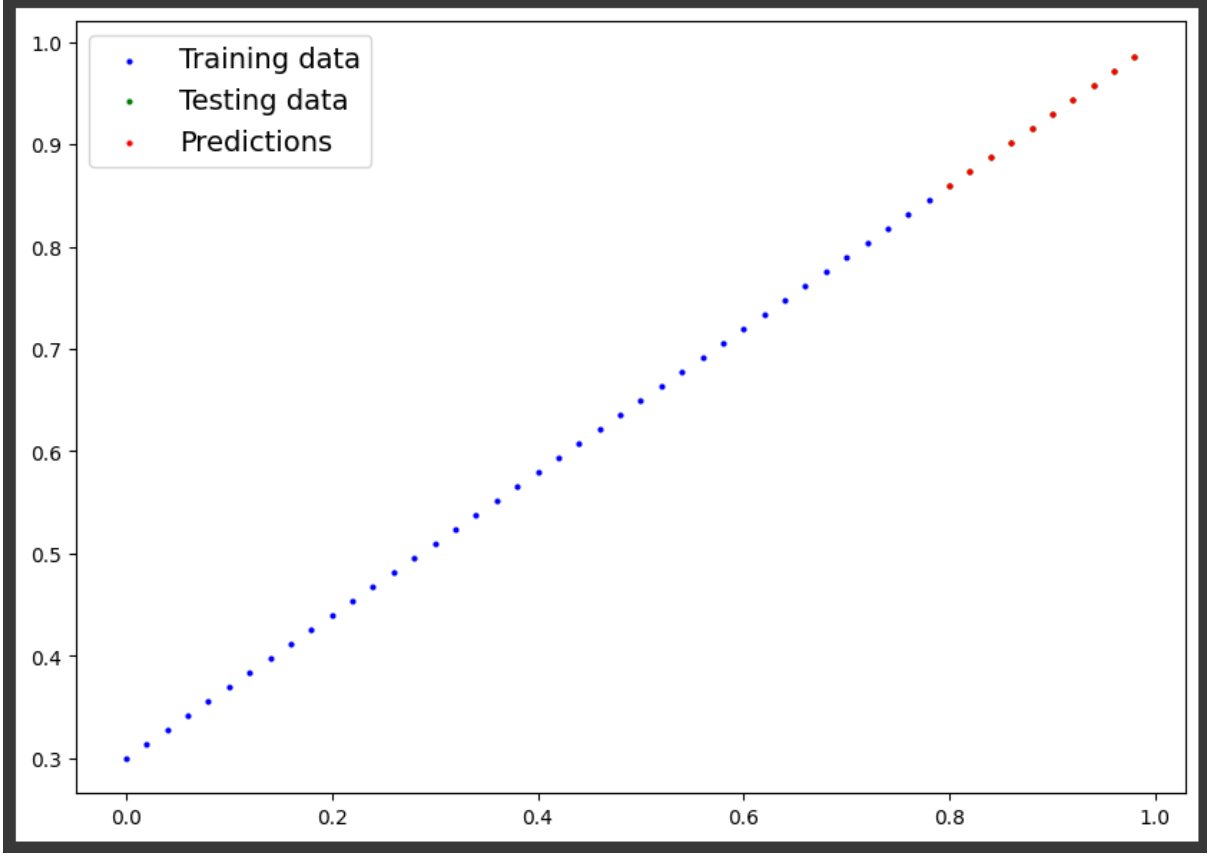
6.4 Making predictions

```
# 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
```

```
tensor([[0.8600],
        [0.8739],
        [0.8878],
        [0.9018],
        [0.9157],
        [0.9296],
        [0.9436],
        [0.9575],
        [0.9714],
        [0.9854]], device='cuda:0')
```

Kode program melakukan prediksi yang dihasilkan oleh model pada data pengujian (y_preds).

```
# plot_predictions(predictions=y_preds) # -> won't work... data not on CPU

# Put data on the CPU and plot it
plot_predictions(predictions=y_preds.cpu())
```



Kode program melakukan visualisasi data terhadap training data, testing data, dan predictions.

Kode program melakukan :

- Membuat models untuk menyimpan model nantinya.
- Membuat path penyimpanan model dengan menggunakan objek Path.
- Menyimpan state_dict dari model (model_1.state_dict()) ke path yang telah ditentukan menggunakan fungsi torch.save().

- Membuat instance baru dari model LinearRegressionModelV2() dan memuat state_dict dari model yang telah disimpan sebelumnya menggunakan torch.load().
- Memindahkan model (loaded_model_1) ke perangkat target yang telah ditentukan sebelumnya (misalnya, CPU atau GPU) menggunakan metode .to(device).
- Menampilkan output model yang telah dimuat dan menunjukkan perangkat tempat model tersebut berada.

PyTorch Neural Network Classification

Klasifikasi dalam konteks machine learning melibatkan proses memprediksi kelas atau kategori dari suatu data berdasarkan fitur-fitur yang ada. Tujuan utamanya adalah untuk mengklasifikasikan data ke dalam kategori yang tepat.

Dalam machine learning, terdapat beberapa jenis masalah klasifikasi yang umum, yaitu:

- Klasifikasi Biner: Masalah ini melibatkan prediksi apakah data masukan termasuk ke dalam salah satu dari dua kategori yang mungkin. Contohnya adalah memprediksi apakah email adalah spam atau bukan, atau apakah pasien memiliki penyakit tertentu atau tidak.
- Klasifikasi Multikelas: Pada masalah ini, data masukan dapat diklasifikasikan ke dalam lebih dari dua kategori. Contohnya adalah mengklasifikasikan gambar menjadi kategori "kucing", "anjing", atau "burung".
- Klasifikasi Multilabel: Masalah ini melibatkan prediksi beberapa label atau kategori yang dapat diberikan kepada data masukan. Contohnya adalah mengklasifikasikan berita ke dalam beberapa topik seperti "olahraga", "politik", dan "hiburan".

Dalam neural network, arsitektur jaringan untuk masalah klasifikasi umumnya terdiri dari lapisan-lapisan berikut:

- Lapisan Input: Lapisan ini menerima data masukan yang akan diklasifikasikan. Jumlah neuron pada lapisan ini sesuai dengan jumlah fitur pada data masukan.
- Lapisan Tersembunyi (Hidden Layer): Lapisan ini berada di antara lapisan input dan lapisan output. Jumlah lapisan tersembunyi dan jumlah neuron per lapisan tergantung pada kompleksitas masalah klasifikasi. Lapisan tersembunyi dapat menggunakan fungsi aktivasi seperti ReLU (Rectified Linear Unit) untuk memperkenalkan non-linearitas ke dalam jaringan.
- Lapisan Output: Lapisan ini menghasilkan prediksi kelas atau kategori dari data masukan. Jumlah neuron pada lapisan ini tergantung pada jenis masalah klasifikasi yang dihadapi. Misalnya, pada klasifikasi biner, lapisan output akan memiliki satu neuron untuk masing-masing kelas yang mungkin, sedangkan pada klasifikasi multikelas, jumlah neuron pada lapisan output sesuai dengan jumlah kelas yang ada.
- Aktivasi Lapisan Tersembunyi: Fungsi aktivasi yang umum digunakan pada lapisan tersembunyi adalah ReLU (Rectified Linear Unit), tetapi ada banyak fungsi aktivasi lain yang dapat digunakan. Fungsi aktivasi pada lapisan tersembunyi tidak memiliki perbedaan signifikan antara klasifikasi biner dan multikelas.
- Aktivasi Lapisan Output: Pada klasifikasi biner, fungsi aktivasi yang umum digunakan pada lapisan output adalah sigmoid (torch.sigmoid dalam PyTorch). Pada

klasifikasi multikelas, fungsi aktivasi yang umum digunakan pada lapisan output adalah softmax (torch.softmax dalam PyTorch).

- Fungsi Kerugian: Pada klasifikasi biner, fungsi kerugian yang umum digunakan adalah binary crossentropy (torch.nn.BCELoss dalam PyTorch). Pada klasifikasi multikelas, fungsi kerugian yang umum digunakan adalah cross entropy (torch.nn.CrossEntropyLoss dalam PyTorch).
- Optimizer: Pada kedua jenis klasifikasi, algoritma optimasi yang dapat digunakan adalah SGD (stochastic gradient descent) dan Adam. Terdapat opsi lain yang dapat ditemukan dalam modul torch.optim.

Dalam klasifikasi, bersama dengan regresi (memprediksi angka), merupakan salah satu jenis masalah machine learning yang paling umum. Pada PyTorch Neural Network Classification ini akan bekerja dengan beberapa masalah klasifikasi menggunakan PyTorch. Disini yang dibahas adalah beberapa hal berikut:

- Architecture of a classification neural network: Neural network dapat memiliki berbagai bentuk dan ukuran, tetapi umumnya mengikuti pola yang serupa.
- Getting binary classification data ready: Data dapat berupa apa saja, tetapi untuk memulai kita akan membuat dataset klasifikasi biner sederhana.
- Building a PyTorch classification model: Membuat model untuk mempelajari pola dalam data, memilih fungsi kerugian, pengoptimal, dan membangun proses pelatihan yang khusus untuk klasifikasi.
- Fitting the model to data (training): Setelah memiliki data dan model, model tersebut akan mencari pola dalam data pelatihan.
- Making predictions and evaluating a model (inference): Menemukan pola dalam data dan membandingkan hasilnya dengan data aktual yang diuji.
- Improving a model (from a model perspective): Setelah melatih dan mengevaluasi model, jika masih belum berfungsi dengan baik, model akan mencoba beberapa cara untuk meningkatkannya.
- Non-linearity: Memodelkan garis lurus.
- Replicating non-linear functions: Menggunakan fungsi non-linear untuk membantu memodelkan data non-linear.
- Putting it all together with multi-class classification: Mengabungkan semua yang telah dilakukan sejauh ini untuk klasifikasi biner dengan masalah klasifikasi multikelas.

PyTorch Computer Vision

PyTorch Computer Vision adalah bidang penggunaan PyTorch yang berkaitan dengan pemrosesan dan analisis gambar serta video menggunakan teknik-teknik komputer visi. Berikut adalah penjelasan lebih rinci mengenai masing-masing masalah tersebut:

- Klasifikasi Biner (Binary Classification): Dalam klasifikasi biner, tujuannya adalah untuk memisahkan objek ke dalam dua kelas yang berbeda. Contohnya adalah membangun model yang dapat mengklasifikasikan apakah sebuah foto adalah kucing atau anjing. Model ini akan belajar membedakan ciri-ciri visual yang membedakan kucing dan anjing, dan kemudian dapat digunakan untuk memprediksi kelas dari foto-foto yang belum pernah dilihat sebelumnya.

- Klasifikasi Multikelas (Multi-class Classification): Klasifikasi multikelas melibatkan memisahkan objek ke dalam lebih dari dua kelas yang berbeda. Misalnya, Anda dapat membangun model yang dapat mengklasifikasikan apakah sebuah foto adalah kucing, anjing, atau ayam. Model ini akan mempelajari perbedaan visual antara ketiga kelas ini dan dapat digunakan untuk mengklasifikasikan foto-foto berdasarkan kelas-kelas tersebut.
- Deteksi Objek (Object Detection): Deteksi objek melibatkan identifikasi dan penempatan kotak pembatas (bounding box) pada objek-objek yang ada dalam gambar atau video. Sebagai contoh, Anda dapat membangun model yang dapat mengidentifikasi lokasi mobil dalam setiap frame video. Model ini akan belajar untuk mengenali ciri-ciri visual mobil dan menghasilkan kotak pembatas yang menunjukkan lokasi mobil dalam setiap frame video.
- Segmentasi Panoptik (Panoptic Segmentation): Segmentasi panoptik adalah tugas yang melibatkan pemisahan setiap objek dalam gambar ke dalam segmen-segmen yang berbeda. Misalnya, Anda dapat membangun model yang dapat memisahkan objek-objek dalam gambar menjadi segmen-segmen yang berbeda. Model ini akan mempelajari ciri-ciri visual yang membedakan objek-objek tersebut dan menghasilkan segmentasi yang memisahkan setiap objek dalam gambar.

Disini yang dibahas adalah beberapa hal berikut:

- Computer vision libraries in PyTorch: PyTorch memiliki beberapa pustaka komputer visi yang berguna yang sudah tersedia, mari kita jelajahi pustaka-pustaka tersebut.
- Load data: Untuk berlatih dalam komputer visi, kita akan memulai dengan menggunakan beberapa gambar pakaian yang berbeda dari FashionMNIST.
- Prepare data: Kita memiliki beberapa gambar, mari kita muat menggunakan PyTorch DataLoader sehingga kita dapat menggunakannya dalam loop pelatihan.
- Model 0: Building a baseline model: Kita akan membuat model klasifikasi multikelas untuk mempelajari pola-pola dalam data. Kita juga akan memilih fungsi loss, optimizer, dan membangun loop pelatihan.
- Making predictions and evaluting model 0: Kita buat beberapa prediksi menggunakan model dasar kita dan mengevaluasinya.
- Setup device agnostic code for future models: Menulis kode yang tidak bergantung pada perangkat adalah praktik terbaik, jadi mari kita persiapkan kode tersebut.
- Model 1: Adding non-linearity: Eksperimen adalah bagian penting dari machine learning, digunakan untuk meningkatkan model dasar dengan menambahkan lapisan non-linear.
- Model 2: Convolutional Neural Network (CNN): Fokus pada komputer visi dan memperkenalkan arsitektur Convolutional Neural Network (CNN).
- Comparing our models: Membangun tiga model yang berbeda.
- Evaluating our best model: Membuat beberapa prediksi pada gambar-gambar acak dan mengevaluasi model terbaik.
- Making a confusion matrix: Mengevaluasi model klasifikasi.
- Saving and loading the best performing model: Model disimpan untuk dimuat kembali.

PyTorch Custom Datasets

Dalam machine learning dengan PyTorch, kita sering perlu menggunakan custom dataset. Dataset kustom adalah kumpulan data yang terkait dengan masalah spesifik yang sedang kita kerjakan. Misalnya, custom dataset dapat berisi gambar makanan untuk aplikasi klasifikasi gambar makanan, atau ulasan pelanggan dengan penilaiannya untuk membangun model klasifikasi ulasan. PyTorch menyediakan library seperti TorchVision, TorchText, TorchAudio, dan TorchRec yang memiliki fungsi-fungsi bawaan untuk memuat custom dataset. Namun, jika fungsi-fungsi tersebut tidak mencukupi, kita dapat membuat subclass dari torch.utils.data.Dataset dan menyesuaikannya sesuai kebutuhan.

Dalam program Custom datasets Langkah-langkah yang dilakukan yaitu:

- Importing PyTorch and setting up device-agnostic code: Mengimpor PyTorch dan menyiapkan kode yang bekerja di berbagai perangkat. Lalu akan memuat PyTorch dan mengikuti praktik terbaik untuk menyiapkan kode agar dapat berjalan di berbagai perangkat.
- Get data: Menggunakan dataset kustom yang berisi gambar-gambar pizza, steak, dan sushi.
- Become one with the data (data preparation): Pada awal setiap masalah pembelajaran mesin baru, sangat penting untuk memahami data yang kita miliki.
- Transforming data: Data yang didapatkan tidak langsung dapat digunakan dengan model machine learning. Di sini kita akan melihat beberapa langkah yang dapat kita lakukan untuk mengubah gambar agar siap digunakan dengan model.
- Loading data with ImageFolder (option 1): Memuat data dengan ImageFolder (pilihan 1) PyTorch memiliki banyak fungsi bawaan untuk memuat data dengan format yang umum digunakan. ImageFolder berguna jika gambar-gambar kita memiliki format klasifikasi gambar standar.
- Loading image data with a custom Dataset: Memuat data gambar dengan Dataset kustom. Jika yang terjadi jika PyTorch tidak memiliki fungsi bawaan untuk memuat data yang kita butuhkan akan dibangun subclass kustom dari torch.utils.data.Dataset.
- Other forms of transforms (data augmentation): Transformasi data lainnya (augmentasi data). Augmentasi data adalah teknik umum untuk memperluas keberagaman data latih.
- Model 0: TinyVGG without data augmentation: Membangun model yang dapat mempelajarinya dan juga akan membuat fungsi-fungsi untuk melatih dan mengevaluasi model kita.
- Exploring loss curves: Mengeksplorasi kurva loss Kurva loss merupakan cara yang baik untuk melihat bagaimana model kita belajar dan memperbaiki performanya seiring waktu.
- Model 1: TinyVGG with data augmentation: TinyVGG dengan augmentasi data Setelah mencoba model tanpa augmentasi, di tahap ini akan dilakukan percobaan menggunakan augmentasi data.
- Compare model results: Membandingkan hasil model Mari kita membandingkan kurva loss dari model-model yang berbeda dan melihat model mana yang memberikan performa terbaik, serta membahas beberapa opsi untuk meningkatkan performa model.

- Making a prediction on a custom image: Melakukan prediksi pada gambar kustom Model kita dilatih menggunakan dataset gambar pizza, steak, dan sushi.

PyTorch Going Modular

PyTorch Going Modular yaitu melibatkan mengubah kode notebook (dari Jupyter Notebook atau notebook Google Colab) menjadi serangkaian skrip Python yang menawarkan fungsionalitas yang serupa. Misalnya, kita bisa mengubah kode notebook kita dari serangkaian sel menjadi file Python berikut:

- data_setup.py - file untuk mempersiapkan dan mengunduh data jika diperlukan.
- engine.py - file yang berisi berbagai fungsi pelatihan.
- model_builder.py atau model.py - file untuk membuat model PyTorch.
- train.py - file untuk memanfaatkan semua file lain dan melatih model PyTorch yang ditargetkan.
- utils.py - file yang didedikasikan untuk fungsi utilitas yang membantu.

Mengapa orang-orang memilih untuk menggunakan pendekatan modular dengan menggunakan python yaitu:

- Reproduksibilitas: Dalam skrip Python, Anda dapat secara eksplisit menentukan semua dependensi dan lingkungan yang diperlukan untuk menjalankan kode. Ini membuatnya lebih mudah untuk mereproduksi lingkungan yang sama di berbagai sistem dan menjaga konsistensi dalam menjalankan kode.
- Kemudahan Penggunaan: Skrip Python dapat dieksekusi langsung dari baris perintah tanpa harus membuka notebook terpisah. Ini memungkinkan pengguna lain atau anggota tim untuk dengan mudah menjalankan dan menggunakan kode yang Anda buat.
- Versioning: Dalam pengembangan perangkat lunak yang lebih besar, menggunakan sistem versioning seperti Git untuk melacak perubahan dan kolaborasi menjadi lebih mudah dengan menggunakan skrip Python. Versi kode dapat diatur dan dikelola dengan lebih baik.
- Skalabilitas: Dalam proyek-proyek yang lebih besar yang melibatkan banyak file dan komponen, menggunakan skrip Python memungkinkan Anda untuk mengemas kode bersama secara terstruktur. Ini menghindari duplikasi kode dan memudahkan pengelolaan dan pemeliharaan proyek yang lebih besar.

PyTorch Transfer Learning

Transfer learning merupakan teknik yang kuat dalam dunia deep learning yang memungkinkan kita untuk mengambil pola (disebut juga bobot) yang telah dipelajari oleh model lain dari masalah lain dan menggunakannya untuk masalah kita sendiri. Misalnya, kita dapat mengambil pola yang telah dipelajari oleh model computer vision dari dataset seperti ImageNet (jutaan gambar objek yang berbeda) dan menggunakannya untuk memperkuat model FoodVision Mini kita. Atau kita bisa mengambil pola dari model bahasa (model yang

telah melalui teks dalam jumlah besar untuk mempelajari representasi bahasa) dan menggunakannya sebagai dasar model untuk mengklasifikasikan sampel teks yang berbeda.

Ada dua manfaat utama dalam menggunakan transfer learning:

- Dapat memanfaatkan model yang sudah ada (biasanya berupa arsitektur jaringan saraf) yang terbukti berhasil dalam masalah yang serupa dengan masalah kita.
- Dapat memanfaatkan model yang sudah belajar pola pada data yang mirip dengan data kita sendiri. Hal ini seringkali menghasilkan hasil yang bagus dengan menggunakan jumlah data kustom yang lebih sedikit.

Pada implementasi kode programnya yaitu menggunakan model pretrained dari torchvision.models dan menyesuaikannya agar dapat digunakan untuk masalah FoodVision Mini. Awal yang dilakukan yaitu:

- Getting setup: Menyiapkan kode program yang sudah disiapkan.
- Get data: Mengambil dataset klasifikasi gambar pizza, steak, dan sushi yang telah digunakan untuk mencoba meningkatkan hasil model.
- Create Datasets and DataLoaders: menggunakan skrip data_setup.py.
- Get and customise a pretrained model: Mendownload model pretrained dari torchvision.models dan menyesuaikannya dengan masalah kita sendiri.
- Train model: Melakukan pelatihan model.
- Evaluate the model by plotting loss curves: Mengevaluasi model, apakah model tersebut overfitting atau underfitting.
- Make predictions on images from the test set: Melakukan metrik evaluasi model dan memvisualisasi prediksi pada sampel-sampel uji.

PyTorch Experiment Tracking

Experiment Tracking merupakan proses mencatat dan mencatat informasi penting tentang setiap eksperimen yang Anda jalankan. Ini melibatkan menyimpan parameter model, metrik evaluasi, hiperparameter, dataset yang digunakan, catatan pelatihan, dan hasil lainnya yang relevan. Pelacakan eksperimen membantu Anda mengatur dan membandingkan eksperimen secara efisien. Experiment Tracking sangat penting dalam machine learning karena membantu menganalisis dan mencatat hasil dari berbagai kombinasi data, arsitektur model, dan regime pelatihan. Dengan experiment tracking kita dapat memahami faktor-faktor apa yang berkontribusi terhadap kinerja model dan membuat keputusan yang lebih baik dalam pengembangan model yang lebih baik.

Beberapa cara yang berbeda untuk experiment tracking dalam machine learning:

- Python dictionaries, CSV files, print outs: Cara yang sederhana untuk mencatat informasi eksperimen. Namun, sulit untuk melacak banyak eksperimen dengan efisien.
- TensorBoard: Cara lain untuk melacak eksperimen. Meskipun mudah diinstal dan dikenal luas pengalaman pengguna tidak sebaik opsi lainnya.

- Weights & Biases Experiment Tracking: Memberikan pengalaman pengguna. Disini kita dapat membuat eksperimen publik dan melacak hampir semua hal. Namun, ini membutuhkan sumber daya eksternal di luar PyTorch.
- MLFlow: Metode lain yang menyediakan manajemen siklus hidup MLOps yang sepenuhnya open-source.

Pada implementasi program Experiment Tracking yaitu:

- Getting setup: Mengunduh dan memastikan kita dapat menggunakan kode yang telah kita tulis sebelumnya.
- Get data: Mendapatkan dataset klasifikasi gambar pizza, steak, dan sushi yang telah kita gunakan sebelumnya untuk mencoba meningkatkan hasil model FoodVision Mini.
- Create Datasets and DataLoaders: menggunakan script data_setup.py.
- Get and customise a pretrained model: Mengunduh model pra-pelatihan dari torchvision.models dan menyesuaikannya.
- Train model amd track results: Melatih dan melacak hasil pelatihan dari satu model menggunakan TensorBoard.
- View our model's results in TensorBoard: Memvisualisasikan kurva kerugian.
- Creating a helper function to track experiments: Membuat fungsi yang akan membantu kita menyimpan hasil eksperimen pemodelan kita.
- Setting up a series of modelling experiments: Menjalankan eksperimen satu per satu, bagaimana jika kita menulis kode untuk menjalankan beberapa eksperimen sekaligus, dengan model yang berbeda, jumlah data yang berbeda, dan waktu pelatihan yang berbeda.
- View modelling experiments in TensorBoard: Menjalankan delapan eksperimen pemodelan sekaligus.
- Load in the best model and make predictions with it: Mengetahui model mana yang memiliki kinerja terbaik.

PyTorch Paper Replicating

Paper replicating merupakan proses menghasilkan kode yang dapat digunakan untuk mereplikasi kemajuan dalam bidang pembelajaran mesin yang terdokumentasikan dalam makalah penelitian. Dalam paper replicating, kita mencoba mengimplementasikan teknik-teknik dan model yang dijelaskan dalam makalah tersebut ke dalam kode yang dapat kita gunakan untuk memecahkan masalah kita sendiri.

Pada implementasi program Paper Replicating yaitu:

- Getting setup: mengunduh dan memastikan bahwa kita dapat menggunakan kode yang telah ditulis sebelumnya.
- Get data: Mendapatkan dataset klasifikasi gambar pizza, steak, dan sushi yang telah kita gunakan sebelumnya, dan membangun sebuah Vision Transformer untuk mencoba meningkatkan hasil model FoodVision Mini.
- Create Datasets and DataLoaders: Menggunakan skrip data_setup.py.

- Replicating the ViT paper: an overview: Proses mereplikasi sebuah paper replicating maching learning, di mana kita akan memecah makalah ViT menjadi bagian-bagian kecil agar dapat mereplikasinya secara bertahap.
- Equation 1: The Patch Embedding: Arsitektur ViT terdiri dari empat persamaan utama, yang pertama adalah patch dan position embedding, yaitu mengubah gambar menjadi urutan patch yang dapat dipelajari.
- Equation 2: Multi-Head Attention (MSA): Mekanisme self-attention/multi-head self-attention (MSA) merupakan inti dari setiap arsitektur Transformer, termasuk arsitektur ViT yang akan membuat blok MSA menggunakan layer bawaan PyTorch.
- Equation 3: Multilayer Perceptron (MLP): Arsitektur ViT menggunakan multilayer perceptron sebagai bagian dari Transformer Encoder dan lapisan output yang akan mulai membuat MLP untuk Transformer Encoder.
- Creating the Transformer Encoder: Sebuah Transformer Encoder umumnya terdiri dari lapisan MSA (persamaan 2) dan MLP (persamaan 3) yang saling bergantian, dihubungkan dengan residual connections.
- Putting it all together to create ViT: Menggabungkan semua komponen yang diperlukan untuk membuat arsitektur ViT.
- Setting up training code for our ViT model: Menyiapkan program latih untuk model ViT.
- Using a pretrained ViT from torchvision.models: Melatih model besar seperti ViT biasanya membutuhkan banyak data. Karena kita hanya bekerja dengan sejumlah kecil gambar pizza, steak, dan sushi.
- Make predictions on a custom image: menguji model terbaik pada gambar terkenal "pizza-dad".

PyTorch Model Deployment

Model Deployment merupakan proses membuat model machine learning dapat diakses oleh orang atau sesuatu yang lain. Orang lain bisa berarti seseorang yang dapat berinteraksi dengan model dengan cara tertentu. Sebagai contoh, seseorang mengambil foto makanan dengan handphone dan kemudian menggunakan model FoodVision Mini untuk mengklasifikasikannya menjadi pizza, steak, atau sushi. Lalu bisa berupa program lain, aplikasi, atau bahkan model lain yang berinteraksi dengan model machine learning. Jika ingin mengimplementasikan model, ada dua pilihan yang bisa kita pilih:

- Penggunaan on-device adalah kecepatan yang tinggi karena tidak ada data yang harus meninggalkan perangkat, menjaga privasi karena data tidak perlu dikirim keluar perangkat, dan tidak memerlukan koneksi internet. Namun, terdapat keterbatasan daya komputasi, penyimpanan yang terbatas, dan memerlukan keterampilan khusus untuk perangkat tertentu.
- Penggunaan di cloud menawarkan daya komputasi yang hampir tak terbatas, kemampuan untuk menggunakan satu model di mana saja melalui API, dan terhubung dengan ekosistem cloud yang sudah ada. Namun, biaya dapat meningkat jika tidak ada batasan skalabilitas yang diterapkan, prediksi dapat menjadi lebih lambat karena data harus meninggalkan perangkat dan kembali, serta ada kekhawatiran privasi karena data harus dikirim keluar perangkat.

Dalam implementasi kode program Model Deployment yaitu:

- Getting setup: Mengunduh kode yang telah kita tulis sebelumnya agar kita dapat menggunakannya kembali.
- Get data: Mengunduh dataset pizza_steak_sushi_20_percent.zip agar kita dapat melatih model dengan dataset yang sama.
- FoodVision Mini model deployment experiment outline: Menjalankan beberapa eksperimen untuk membandingkan dua model terbaik kita saat ini, yaitu EffNetB2 dan ViT.
- Creating an EffNetB2 feature extractor: Membuat extractor fitur EffNetB2 yang telah berhasil dalam eksperimen sebelumnya sebagai kandidat untuk di-deploy.
- Creating a ViT feature extractor: Membuat extractor fitur ViT yang merupakan model terbaik saat ini untuk dataset pizza, steak, sushi sebagai kandidat untuk di-deploy bersama EffNetB2.
- Making predictions with our trained models and timing them: Melakukan prediksi dengan model yang telah kita buat dan melacak hasilnya.
- Comparing model results, prediction times and size: Membandingkan model untuk melihat model mana yang lebih baik sesuai dengan tujuan.
- Bringing FoodVision Mini to life by creating a Gradio demo: Membuat aplikasi FoodVision Mini to life by creating a Gradio demo.
- Turning our FoodVision Mini Gradio demo into a deployable app: Mempersiapkan untuk demo.
- Deploying our Gradio demo to HuggingFace Spaces: Mendeploy Gradio demo to HuggingFace Spaces untuk dapat diakses oleh semua orang.
- Creating a BIG surprise: Deploy berhasil.
- Deploying our BIG surprise: Mencoba deploy aplikasi lainnya.