TECHNICAL REPORT MACHINE LEARNING

Pytorch Deep Learning

Disusun oleh:

Faishal Anwar

1103204096

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

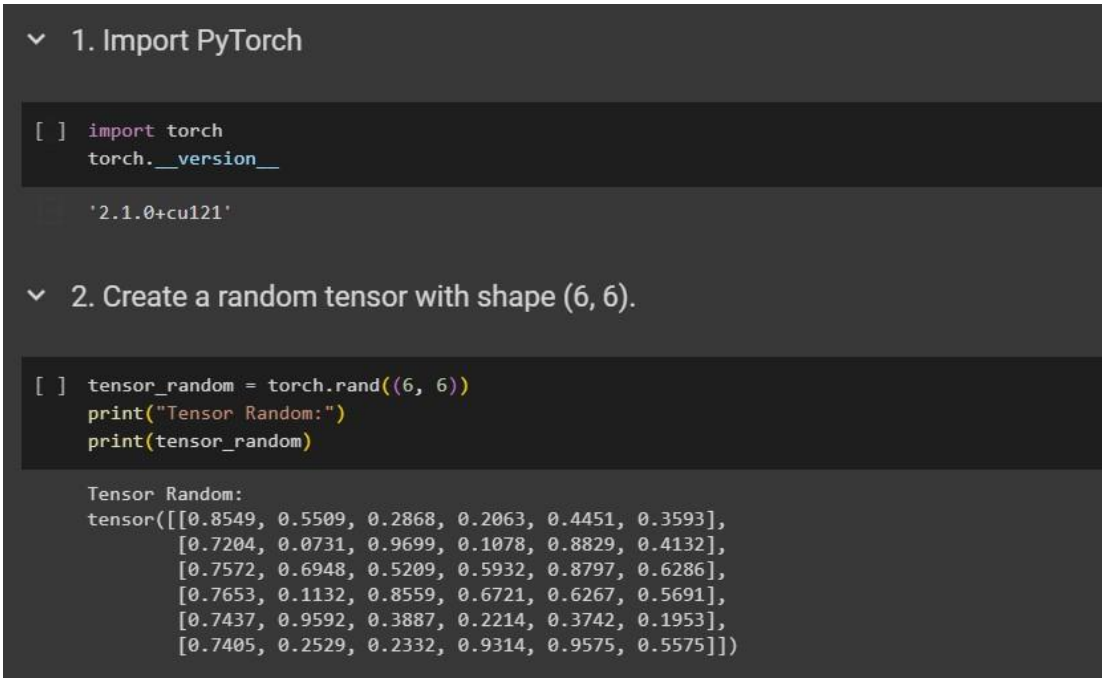
2014

Pendahuluan

PyTorch adalah sebuah pustaka deep learning berbasis Python dan Torch yang dioptimalkan untuk manipulasi tensor. Pustaka ini digunakan untuk membangun aplikasi yang memanfaatkan jaringan neural, memungkinkan data scientist dan pengembang perangkat lunak untuk dengan cepat mengubah arsitektur jaringan. PyTorch memiliki back-end yang berbeda untuk CPU, GPU, dan berbagai fitur fungsional lainnya. Misalnya, untuk CPU, PyTorch menggunakan tensor back-end yang disebut TH, sementara untuk GPU, digunakan back-end yang disebut THC.

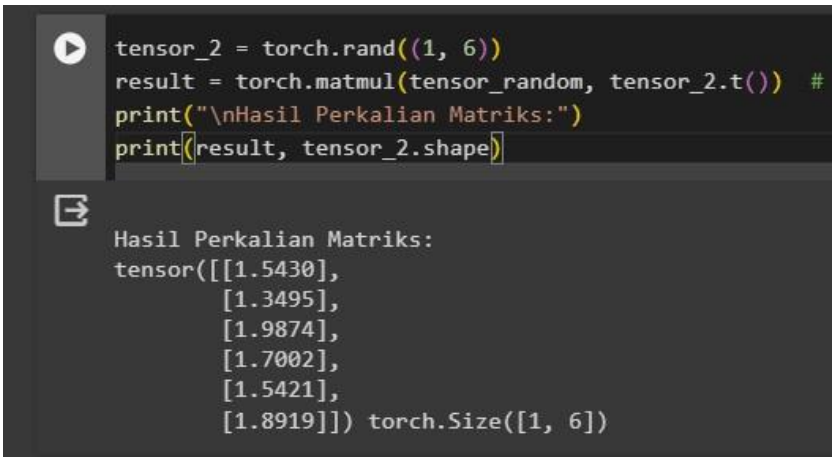
PyTorch dirancang khusus agar mudah dipahami dan digunakan. Dalam konteks PyTorch, tensor berfungsi sebagai struktur data untuk menyimpan informasi numerik, dan mereka mendukung operasi matematika yang efisien. Sebagai contoh, tensor dapat digunakan untuk merepresentasikan input dan output dalam jaringan saraf, dan berbagai operasi tensor dapat diterapkan untuk melakukan perhitungan seperti transformasi linier, aktivasi, dan fungsi matematika lainnya.

1. CHAPTER 00 PyTorch Fundamentals



Baris "import torch" digunakan untuk memasukkan atau mengimpor pustaka PyTorch ke dalam program, memungkinkan penggunaan fungsionalitas PyTorch dalam kode tersebut. Baris berikutnya, "torch.__version__", bertujuan untuk mencetak versi PyTorch yang telah terpasang. Dari hasil keluaran, dapat terlihat bahwa versi yang terinstall adalah '2.1.0+cu121'.

Setelah itu, dibuat sebuah tensor dengan ukuran 6x6, di mana nilainya akan diisi secara acak antara 0 dan 1.

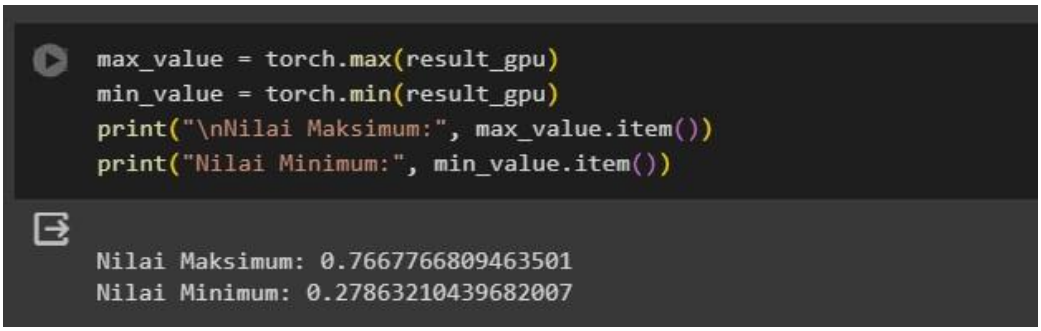


```
tensor_2 = torch.rand((1, 6))
result = torch.matmul(tensor_random, tensor_2.t())  #
print("\nHasil Perkalian Matriks:")
print(result, tensor_2.shape)
```

```
Hasil Perkalian Matriks:
tensor([[1.5430],
        [1.3495],
        [1.9874],
        [1.7002],
        [1.5421],
        [1.8919]]) torch.Size([1, 6])
```

Baris pertama menciptakan sebuah tensor baru dengan dimensi (1, 6), yang kemudian diisi dengan nilai acak. Selanjutnya, hasil dari tensor baru ini akan dioperasikan dengan suatu matriks. Matriks tersebut akan dikalikan dengan tensor yang memiliki dimensi (6x6) yang telah dibuat sebelumnya.

Hasil perkalian ini kemudian akan ditampilkan dalam output kode, dan dimensi tensor yang dihasilkan akan dicetak dengan menggunakan fungsi "torch size" dengan nilai ([1, 6])..



```
max_value = torch.max(result_gpu)
min_value = torch.min(result_gpu)
print("\nNilai Maksimum:", max_value.item())
print("Nilai Minimum:", min_value.item())
```

```
Nilai Maksimum: 0.7667766809463501
Nilai Minimum: 0.27863210439682007
```

Dalam baris program ini, tujuannya adalah untuk mengidentifikasi indeks maksimum (argmax) dan indeks minimum (argmin) dari tensor yang disebut result_gpu. Setelah itu, nilai indeks maksimum dan minimum dari tensor result_gpu tersebut dicetak. Penggunaan .item() di sini dimaksudkan untuk mengambil nilai skalar dari tensor yang hanya memiliki satu elemen. Dengan demikian, perintah ini mencetak indeks maksimum dan minimum sebagai nilai skalar dari tensor tersebut..

```
[ ]  torch.manual_seed(7)
     tensor_random_4d = torch.rand((1, 1, 1, 10))
     tensor_flat = tensor_random_4d.view(-1)
     print("\nTensor Pertama dan Bentuknya:")
     print(tensor_random_4d)
     print(tensor_random_4d.shape)
     print("\nTensor Kedua dan Bentuknya setelah Menghapus Dimensi 1:")
     print(tensor_flat)
     print(tensor_flat.shape)

     Tensor Pertama dan Bentuknya:
     tensor([[[[0.5349, 0.1988, 0.6592, 0.6569, 0.2328, 0.4251, 0.2071, 0.6297,
                0.3653, 0.8513]]]])
     torch.Size([1, 1, 1, 10])

     Tensor Kedua dan Bentuknya setelah Menghapus Dimensi 1:
     tensor([0.5349, 0.1988, 0.6592, 0.6569, 0.2328, 0.4251, 0.2071, 0.6297, 0.3653,
             0.8513])
     torch.Size([10])
```

Pernyataan "torch.manual_seed(7)" di baris kode ini menetapkan nilai benih acak PyTorch menjadi 7. Hal ini bertujuan agar hasil acak dari fungsi-fungsi PyTorch dalam kode tetap konsisten jika benih acak tidak diubah. Pada akhirnya, sebuah tensor dengan empat dimensi akan dicetak, kemudian ukurannya akan diubah dengan menghapus dimensi yang memiliki ukuran 1. Akibatnya, ukuran tensor yang dihasilkan menjadi 10.10.

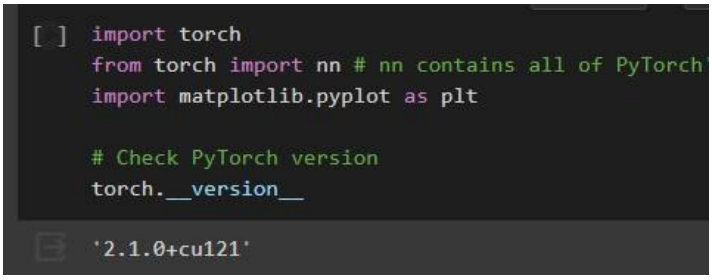
2. CHAPTER 01 PyTorch Workflow Fundamentals

PyTorch Workflow Fundamentals melibatkan serangkaian langkah dalam penggunaan PyTorch untuk proyek deep learning. Berikut adalah beberapa tahapan yang dapat diantisipasi dalam Fundamental PyTorch Workflow:

1. Persiapan Data: Persiapkan data yang diperlukan untuk proyek deep learning. Data ini bisa berupa berkas CSV, gambar, atau video, tergantung pada jenis masalah yang sedang diatasi.
2. Pembangunan Model: Bangun model yang dapat memahami pola-pola dalam data. Model ini bisa berbentuk jaringan saraf, jaringan konvolusi, atau jaringan LSTM. Pengguna dapat menggunakan modul torch.nn untuk merancang dan membangun model ini.
3. Pengujian Model: Latih model menggunakan data yang sudah dipersiapkan sebelumnya. Dalam proses ini, gunakan fungsi kehilangan (loss function) dan optimizer untuk menyesuaikan parameter-model agar sesuai dengan data latihan.
4. Prediksi dan Evaluasi: Setelah pelatihan model, gunakan model untuk membuat prediksi pada data baru. Evaluasi kinerja model dapat dilakukan dengan metrik seperti akurasi, presisi, atau F1-score.

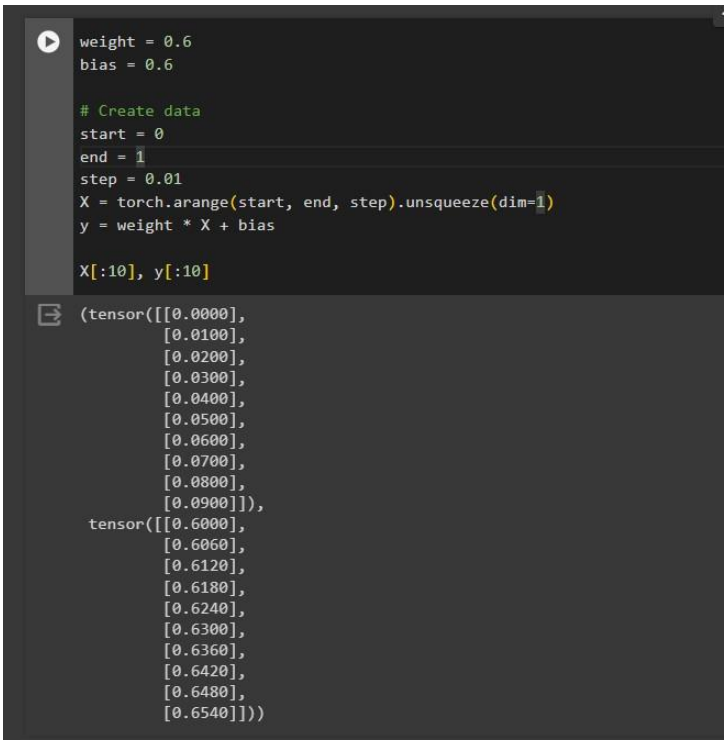
5. Serde (Serialization dan Deserialization) Model: Jika ingin menggunakan model untuk prediksi data baru atau mengoptimalkan model, gunakan teknik serde (serialization) untuk menyimpan model ke file dan mengimpor model ke sistem lain.

Dengan mengikuti langkah-langkah ini, PyTorch digunakan untuk mengimplementasikan model deep learning dan meningkatkan kinerjanya pada tugas tertentu. Pengguna akan memahami cara menggunakan fitur-fitur PyTorch seperti torch.nn, torch.optim, dan torch.nn.functional untuk merancang dan mengoptimalkan model deep learning..



```
[ ] import torch
    from torch import nn # nn contains all of PyTorch
    import matplotlib.pyplot as plt

    # Check PyTorch version
    torch.__version__

'2.1.0+cu121'
```

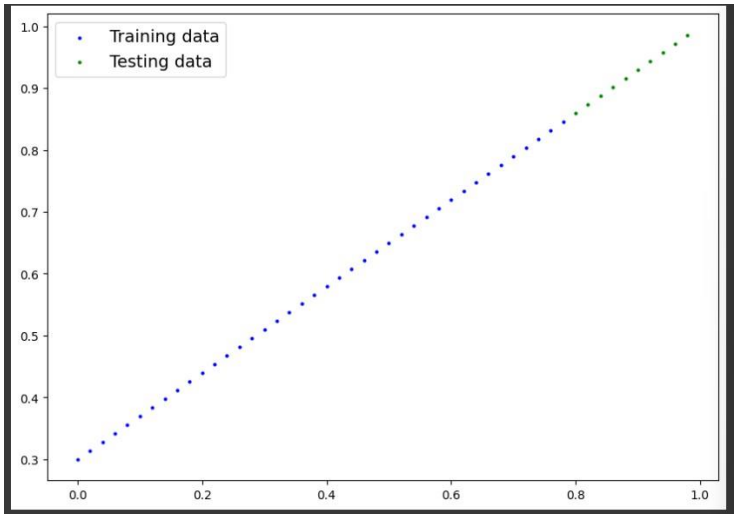
Pada tahap awal, seperti biasanya, dilakukan pemanggilan beberapa pustaka yang diperlukan untuk mengakses dataset. Langkah berikutnya adalah mencetak versi dari pustaka PyTorch itu sendiri..



```
weight = 0.6
bias = 0.6

# Create data
start = 0
end = 1
step = 0.01
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias

X[:10], y[:10]

(tensor([[0.0000],
         [0.0100],
         [0.0200],
         [0.0300],
         [0.0400],
         [0.0500],
         [0.0600],
         [0.0700],
         [0.0800],
         [0.0900]]),
 tensor([[0.6000],
         [0.6060],
         [0.6120],
         [0.6180],
         [0.6240],
         [0.6300],
         [0.6360],
         [0.6420],
         [0.6480],
         [0.6540]]))
```

Pada awal line mendefinisikan bobot (weight) dan bias dengan nilai masing-masing 0.6. membuat data input (X) menggunakan fungsi torch.arange(), yang menghasilkan nilai mulai dari 0 hingga 1 dengan langkah sebesar 0.01. Kemudian, dengan menggunakan metode .unsqueeze(dim=1), menambahkan dimensi tambahan pada tensor X sehingga bentuknya menjadi (100, 1). Selanjutnya membuat data output (y) dengan menggunakan persamaan linear sederhana yaitu y = weight * X + bias. Ini mewakili garis lurus dengan kemiringan (weight) 0.6 dan intersep (bias) 0.6. Terakhir membuat data output (y) dengan menggunakan persamaan linear sederhana yaitu y = weight * X + bias. Ini mewakili garis lurus dengan kemiringan (weight) 0.6 dan intersep (bias) 0.6.

```python
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")
  plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")
  if predictions is not None:
    plt.scatter(test_data, predictions, c="r", s=4, label="Predictions")

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

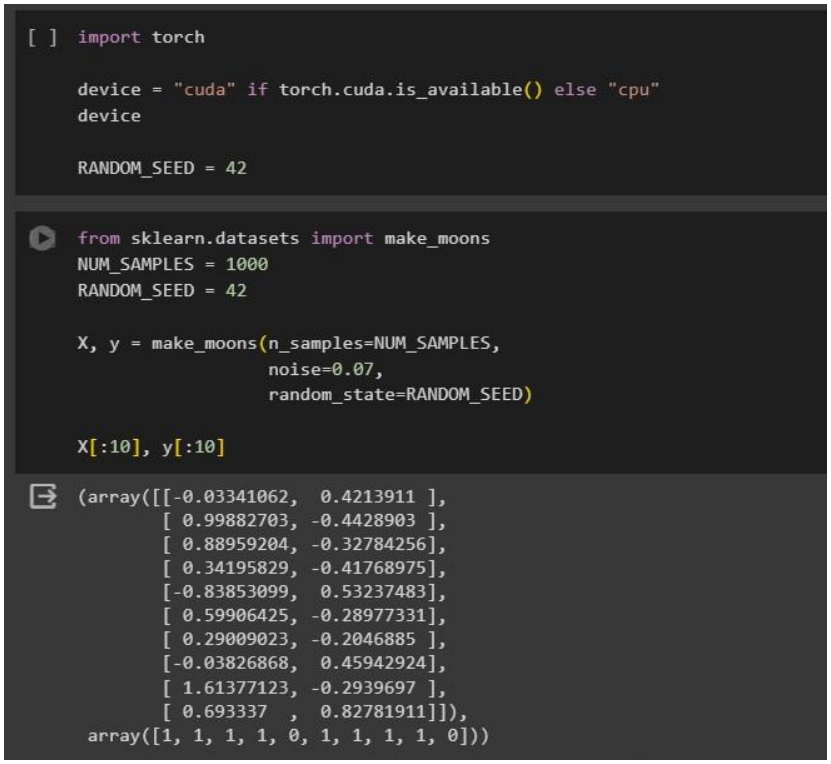


Fungsi `plot_predictions` digunakan untuk membuat visualisasi plot yang menampilkan data latih, data uji, dan perbandingan prediksi (jika ada). Berikut adalah penjelasan singkat mengenai fungsinya:

- **plt.figure(figsize=(10, 7))**: Membuat gambar (figure) dengan ukuran 10x7 inci menggunakan modul matplotlib.
- **plt.scatter(train_data, train_labels, c="b", s=4, label="Training data**"): Membuat scatter plot untuk data latih dengan warna biru ("b"), ukuran titik 4, dan memberikan label "Training data".
- **plt.scatter(test_data, test_labels, c="g", s=4, label="Testing data")** :Membuat scatter plot untuk data uji dengan warna hijau ("g"), ukuran titik 4, dan memberikan label "Testing data".
- **if predictions is not None** : Memeriksa apakah terdapat prediksi yang diberikan. Jika ada, maka baris-baris di bawahnya akan dijalankan.
- plt**.scatter(test_data, predictions, c="r", s=4, label="Predictions")**: Membuat scatter plot untuk prediksi dengan warna merah ("r"), ukuran titik 4, dan memberikan label "Predictions".
- **plt.legend(prop={"size": 14}):** Menambahkan legenda ke dalam plot dengan mengatur ukuran teks legenda menjadi 14. Legenda ini memberikan informasi tentang elemen-elemen yang diplot, seperti data latih, data uji, dan prediksi (jika ada). Terakhir, data prediksi divisualisasikan.

CHAPTER 002 "PyTorch Neural Network Classification"

PyTorch Neural Network Classification mencakup penerapan PyTorch untuk membangun dan melatih jaringan saraf dalam konteks klasifikasi. Langkah-langkah umum dalam proses klasifikasi menggunakan PyTorch melibatkan pembuatan dataset kustom, konstruksi jaringan saraf, pelatihan jaringan saraf, dan evaluasi kinerja model. Proses ini melibatkan penggunaan modul PyTorch seperti torch.nn untuk merancang model, torch.optim untuk mengelola optimizer, dan fungsi kehilangan (loss function) untuk mengukur performa model. Selain itu, tahap-tahap ini juga mencakup persiapan data, pengujian model, dan pengiriman model untuk digunakan pada data baru. Dengan menggunakan PyTorch, pengguna dapat membuat dan mengoptimalkan model klasifikasi jaringan saraf untuk berbagai tugas..



```
[ ] import torch

    device = "cuda" if torch.cuda.is_available() else "cpu"
    device

    RANDOM_SEED = 42
```

```
from sklearn.datasets import make_moons
NUM_SAMPLES = 1000
RANDOM_SEED = 42

X, y = make_moons(n_samples=NUM_SAMPLES,
                  noise=0.07,
                  random_state=RANDOM_SEED)

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

```
(array([[-0.03341062,  0.4213911 ],
        [ 0.99882703, -0.4428903 ],
        [ 0.88959204, -0.32784256],
        [ 0.34195829, -0.41768975],
        [-0.83853099,  0.53237483],
        [ 0.59906425, -0.28977331],
        [ 0.29009023, -0.2046885 ],
        [-0.03826868,  0.45942924],
        [ 1.61377123, -0.2939697 ],
        [ 0.693337  ,  0.82781911]]),
 array([1, 1, 1, 1, 0, 1, 1, 1, 1, 0]))
```

Kode ini memanfaatkan fungsi `make_moons` dari scikit-learn untuk membentuk sebuah dataset yang terdiri dari dua set data dengan pola yang menyerupai bentuk bulan sabit. Berikut adalah penjelasan ringkas mengenai kode tersebut:

- **from sklearn.datasets import make_moons** : Mengimpor fungsi `make_moons` dari modul `datasets` di pustaka scikit-learn.
- **NUM_SAMPLES = 1000** : Menetapkan jumlah sampel yang ingin dihasilkan sebanyak 1000.
- **RANDOM_SEED = 42**: Menetapkan nilai benih acak (random seed) pada 42 untuk memastikan reproduktibilitas hasil.
- **X ,y =** make_moons(n_samples=NUM_SAMPLES, noise=0.07, random_state=RANDOM_SEED)`: Menggunakan fungsi `make_moons` untuk membuat dataset dengan parameter:
  - **n_samples=NUM_SAMPLES**: Menentukan jumlah sampel yang diinginkan sesuai dengan nilai yang telah ditetapkan sebelumnya.
  - **noise=0.07**: Menetapkan tingkat kebisingan pada data sebesar 0.07.
  - **random_state =RANDOM_SEED**`: Menetapkan benih acak untuk memastikan hasil yang konsisten.

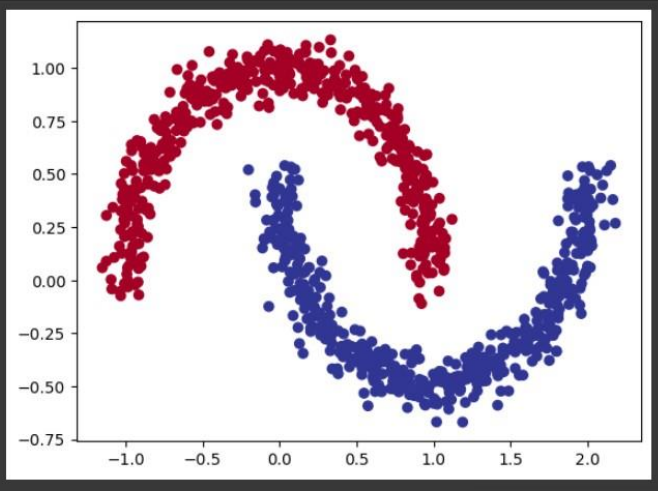
Hasilnya, variabel X akan berisi array dua dimensi yang mewakili koordinat dari setiap sampel, sementara variabel y akan berisi array satu dimensi yang memuat label kelas untuk setiap sampel (0 atau 1), sesuai dengan pola bentuk bulan sabit yang dihasilkan oleh fungsi `make_moons`. Sebagai contoh, dalam kode ini, ditampilkan 10 sampel pertama dari X dan y

```
import pandas as pd
data_df = pd.DataFrame({"X0": X[:, 0],
                        "X1": X[:, 1],
                        "y": y})
data_df.head()
```

	X0	X1	y
0	-0.033411	0.421391	1
1	0.998827	-0.442890	1
2	0.889592	-0.327843	1
3	0.341958	-0.417690	1
4	-0.838531	0.532375	0

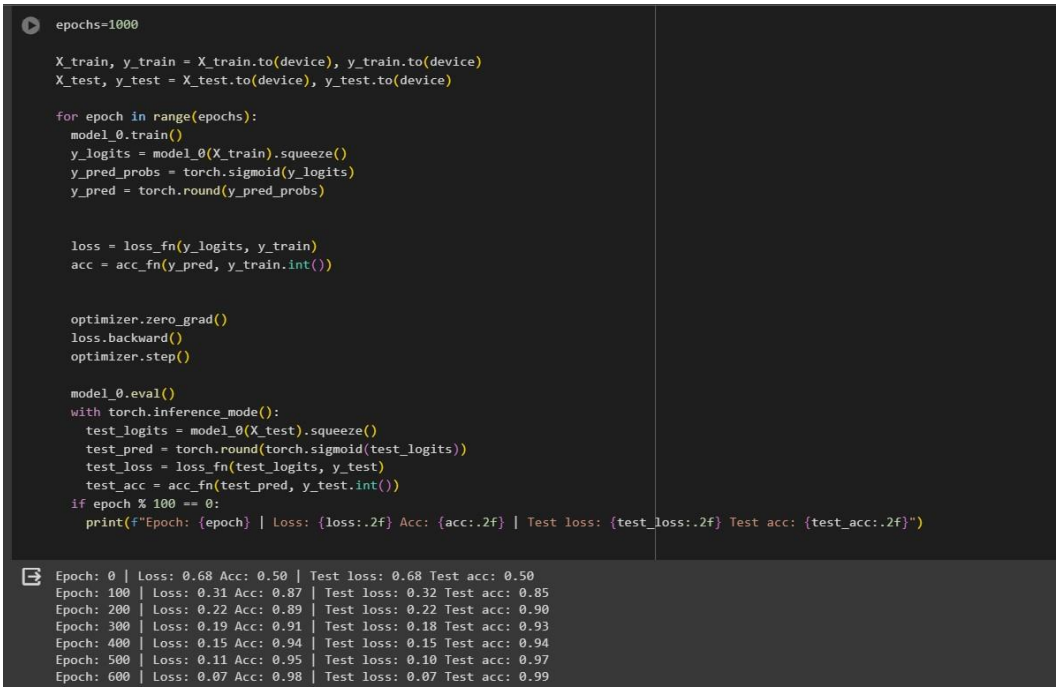
Dilakukan pembuatan DataFrame menggunakan Pandas (pd) dengan perintah `data_df = pd.DataFrame({"X0": X[:, 0], "X1": X[:, 1], "y": y})`. Pada DataFrame ini, kolom pertama ("X0") diisi dengan nilai dari kolom pertama array X (X[:, 0]), kolom kedua ("X1") diisi dengan nilai dari kolom kedua array X (X[:, 1]), dan kolom terakhir ("y") diisi dengan nilai dari array y. Selanjutnya, dilakukan pencetakan lima baris pertama dari DataFrame yang baru dibuat menggunakan perintah `data_df.head()`. Penggunaan metode .head() bertujuan untuk memberikan gambaran singkat mengenai struktur dan isi DataFrame tersebut. Secara rinci, DataFrame data_df terdiri dari tiga kolom, yaitu "X0" dan "X1" yang memuat koordinat dari setiap sampel, dan "y" yang berisi label kelas (0 atau 1) untuk setiap sampel. Pendekatan ini memberikan cara yang lebih terstruktur untuk menyimpan dan mengelola dataset jika dibandingkan dengan menggunakan array numpy secara langsung..

```
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu);
```



Dengan menggunakan perintah import matplotlib.pyplot as plt, dilakukan impor library matplotlib dan memberikan alias sebagai plt. Selanjutnya, perintah plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu) digunakan untuk membuat scatter plot menggunakan matplotlib. Pada plot ini, X[:, 0] dan X[:, 1] digunakan sebagai koordinat sumbu x dan y, sedangkan c=y menentukan warna untuk setiap titik berdasarkan label kelas yang terdapat dalam array y. Parameter cmap=plt.cm.RdYlBu digunakan untuk menentukan peta warna yang akan digunakan, dimulai dari merah, melalui kuning, hingga biru.

Akibatnya, scatter plot yang dihasilkan menampilkan setiap titik sebagai representasi dari satu sampel, dengan warna titik mencerminkan label kelasnya. Selain itu, posisi titik pada sumbu x dan y merepresentasikan koordinat dari setiap sampel. Visualisasi ini memberikan gambaran yang berguna untuk memahami distribusi dan pola dataset "moons" yang telah dibuat..



```
epochs=1000

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):
  model_0.train()
  y_logits = model_0(X_train).squeeze()
  y_pred_probs = torch.sigmoid(y_logits)
  y_pred = torch.round(y_pred_probs)


  loss = loss_fn(y_logits, y_train)
  acc = acc_fn(y_pred, y_train.int())


  optimizer.zero_grad()
  loss.backward()
  optimizer.step()

  model_0.eval()
  with torch.inference_mode():
    test_logits = model_0(X_test).squeeze()
    test_pred = torch.round(torch.sigmoid(test_logits))
    test_loss = loss_fn(test_logits, y_test)
    test_acc = acc_fn(test_pred, y_test.int())
  if epoch % 100 == 0:
    print(f"Epoch: {epoch} | Loss: {loss:.2f} Acc: {acc:.2f} | Test loss: {test_loss:.2f} Test acc: {test_acc:.2f}")
```

```
Epoch: 0 | Loss: 0.68 Acc: 0.50 | Test loss: 0.68 Test acc: 0.50
Epoch: 100 | Loss: 0.31 Acc: 0.87 | Test loss: 0.32 Test acc: 0.85
Epoch: 200 | Loss: 0.22 Acc: 0.89 | Test loss: 0.22 Test acc: 0.90
Epoch: 300 | Loss: 0.19 Acc: 0.91 | Test loss: 0.18 Test acc: 0.93
Epoch: 400 | Loss: 0.15 Acc: 0.94 | Test loss: 0.15 Test acc: 0.94
Epoch: 500 | Loss: 0.11 Acc: 0.95 | Test loss: 0.10 Test acc: 0.97
Epoch: 600 | Loss: 0.07 Acc: 0.98 | Test loss: 0.07 Test acc: 0.99
```

Kode ini adalah contoh implementasi pelatihan dan evaluasi model menggunakan PyTorch. Jumlah epoch diatur sebanyak 1000. Proses pelatihan dilakukan dengan melakukan loop melalui setiap epoch:

a. model_0.train(): Menetapkan model ke mode pelatihan untuk memungkinkan pembaruan parameter model.

b. Menghitung prediksi dan nilai logit untuk data latih:

- y_logits = model_0(X_train).squeeze()`: Mendapatkan logit (output sebelum fungsi aktivasi) dari model untuk data latih.
- y_pred_probs = torch.sigmoid(y_logits): Menggunakan fungsi sigmoid untuk menghitung probabilitas dari logit.
- y_pred = torch.round(y_pred_probs): Membulatkan probabilitas menjadi nilai biner (0 atau 1).

c. Menghitung nilai loss dan akurasi untuk data latih:

- loss = loss_fn(y_logits, y_train): Menggunakan fungsi loss (sepertinya menggunakan binary cross entropy) untuk menghitung nilai loss.
- acc = acc_fn(y_pred, y_train.int()): Menggunakan fungsi akurasi untuk menghitung nilai akurasi.

d. Proses backpropagation:

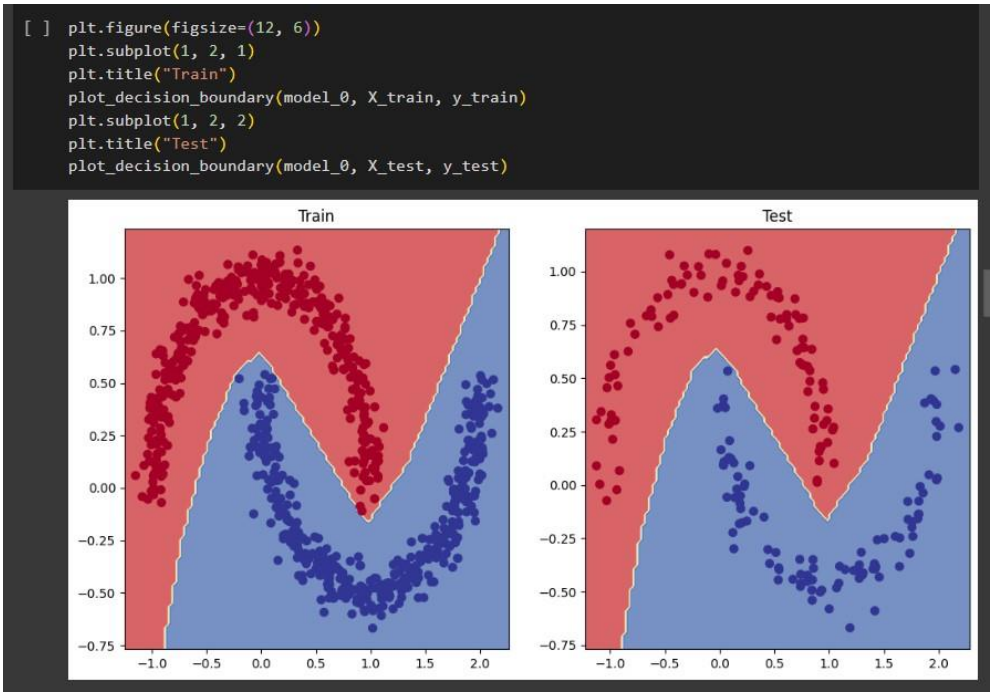
- optimizer.zero_grad(): Mengosongkan gradien dari parameter model.
- loss.backward(): Menghitung gradien loss terhadap parameter model.
- optimizer.step(): Melakukan satu langkah optimisasi untuk mengupdate parameter model.

e. model_0.eval(): Menetapkan model ke mode evaluasi untuk mematikan dropout atau pengaruh lainnya selama evaluasi.

f. Evaluasi model pada data uji:

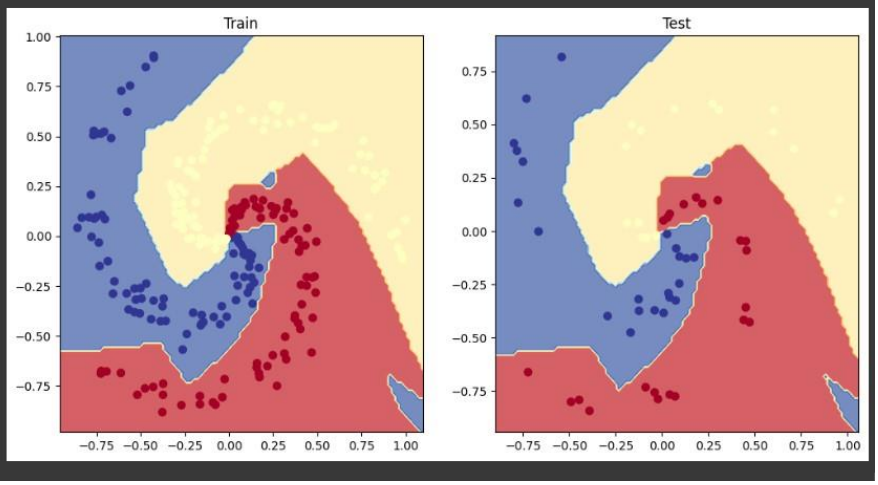
- with torch.inference_mode(): Menggunakan mode inferensi untuk menghindari perhitungan gradien dan menghemat memori.
- test_logits = model_0(X_test).squeeze()`: Mendapatkan logit dari model untuk data uji.
- test_pred = torch.round(torch.sigmoid(test_logits))`: Menghitung prediksi untuk data uji.
- test_loss = loss_fn(test_logits, y_test): Menghitung loss untuk data uji.
- test_acc = acc_fn(test_pred, y_test.int()): Menghitung akurasi untuk data uji.

g. Pada setiap iterasi 100 epoch, hasil pelatihan dan evaluasi dicetak, termasuk nilai loss dan akurasi.asi.



Dengan memanfaatkan dua subplot, kita dapat dengan mudah membandingkan kinerja model pada data latih dan data uji, serta mengamati cara model mengklasifikasikan sampel-sampel pada batas keputusannya. Fungsi `plot_decision_boundary` digunakan untuk menampilkan secara visual batas keputusan model pada data dua dimensi..

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Train")
plot_decision_boundary(model_1, X_train, y_train)
plt.subplot(1, 2, 2)
plt.title("Test")
plot_decision_boundary(model_1, X_test, y_test)
```



Dengan menggunakan dua subplot, kita dapat membandingkan kinerja model_1 pada data latih dan data uji, serta mengamati cara model tersebut mengklasifikasikan sampel-sampel pada batas keputusannya. Fungsi `plot_decision_boundary` digunakan untuk secara visual menampilkan batas keputusan model pada data dua dimensi. Implementasi spesifik dari fungsi tersebut mungkin mengandung logika tertentu yang disesuaikan dengan kode yang digunakan..