TECHNICAL REPORT MACHINE LEARNING

Pytorch Deep Learning

Disusun oleh:

Fahmi Adhiwangsa 1103204142

PROGRAM STUDI TEKNIK KOMPUTER

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

2014

Pendahuluan

PyTorch adalah pustaka tensor deep learning yang dioptimalkan berdasarkan Python dan Torch. PyTorch digunakan untuk aplikasi yang menggunakan jaringan neural dan memungkinkan data scientist dan software developer mengubah jaringan dengan cepat. PyTorch menggunakan back-end yang berbeda untuk CPU, GPU, dan berbagai fitur fungsional lainnya. PyTorch menggunakan tensor back-end TH untuk CPU dan THC untuk GPU. PyTorch dirancang khusus agar intuitif dan mudah digunakan.

Dalam PyTorch, tensor dapat digunakan untuk menyimpan data numerik, dan mereka mendukung operasi matematika yang efisien. Misalnya, tensor dapat digunakan untuk merepresentasikan input dan output dalam jaringan saraf, dan operasi tensor dapat diterapkan untuk melakukan perhitungan yang melibatkan transformasi linier, aktivasi, dan fungsi lainnya.

1. CHAPTER 00 PyTorch Fundamentals

1. Import PyTorch

```
[ ] import torch
    torch.__version__
```

```
'2.1.0+cu121'
```

2. Create a random tensor with shape (6, 6).

```
[ ] tensor_random = torch.rand((6, 6))
    print("Tensor Random:")
    print(tensor_random)
```

```
Tensor Random:
tensor([[0.8549, 0.5509, 0.2868, 0.2063, 0.4451, 0.3593],
        [0.7204, 0.0731, 0.9699, 0.1078, 0.8829, 0.4132],
        [0.7572, 0.6948, 0.5209, 0.5932, 0.8797, 0.6286],
        [0.7653, 0.1132, 0.8559, 0.6721, 0.6267, 0.5691],
        [0.7437, 0.9592, 0.3887, 0.2214, 0.3742, 0.1953],
        [0.7405, 0.2529, 0.2332, 0.9314, 0.9575, 0.5575]])
```

Pada baris **import torch** bertujuan untuk mengimpor library PyTorch, memungkinkan untuk menggunakan fungsionalitas PyTorch dalam sebuah program. Selanjutnya pada baris kedua yaitu **torch.__version__** yaitu untuk mencetak versi dari library PyTorch yang terinstal, kita bisa melihat dari outputnya bahwa versi yang terinstal adalah versi **‘2.1.0+cu121’.** Kemudian dibuatkan sebuahkanlah tendor dengan ukuran 6x6, yang nanti isinya akan berniali 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])
```

Pada baris awal, dibuat untuk membuat tendor baru dengan ukuran (1,6) yang nantinya akan diisi dengan acak juga. Selajutnya hasil tendor baru akan dikalikan dengan matriks, yang nantinya akan dikalikan dengan tendor yang berukuran (6x6) yang telah dibuat sebelumnya. Lalu hasil dari perkalian tersebut akan ditampilkan pada outpot codingan, dengan torch size ([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
```

Pada line program ini untuk menemukan indeks maksimum (argmax) dan indeks minimum (argmin) dari tensor **result_gpu**. Mencetak indeks maksimum dan minimum dari tensor **result_gpu. .item()** digunakan untuk mendapatkan nilai skalar dari tensor yang memiliki hanya satu elemen. Jadi, ini mencetak indeks maksimum dan minimum sebagai nilai skalar.

```
[ ]  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])
```

torch.manual_seed(7) baris ini mengatur benih acak PyTorch menjadi 7, sehingga hasil-hasil acak dari fungsi-fungsi PyTorch dalam kode ini akan konsisten jika benih acak tidak berubah. Pada hasil akhir tensor akan dicetak 4 dimensi selanjutnya tensor akan diubuah ukuranya dengan menghapus dimensi dengan ukuran 1. Maka didapatkan torch_size berukuran 10.

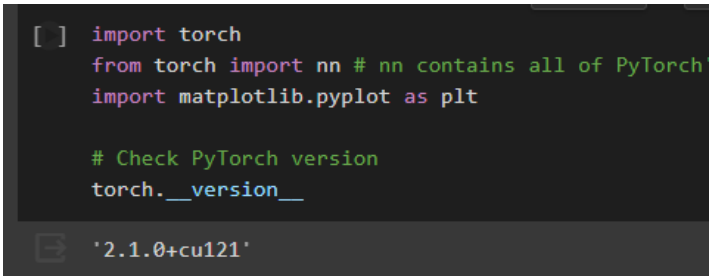
2. CHAPTER 01 PyTorch Workflow Fundamentals

PyTorch Workflow Fundamentals mencakup pengalaman kerja str untuk menggunakan PyTorch dalam proyek deep learning. Berikut ini adalah beberapa langkah yang dapat diperkirakan dari PyTorch Workflow Fundamentals:

- **Persiapan data**: Persiapkan data yang akan digunakan dalam proyek deep learning. Data dapat berupa file CSV, gambar, atau video, tergantung pada masalah yang ingin latih.
- **Membangun model**: Membuat model yang akan mempelajari pola dalam data. Model ini dapat berupa jaringan neuron, seperti jaringan saraf, jaringan konvolusi, atau jaringan LSTM. dapat menggunakan torch.nn untuk membangun model ini.
- **Pengujian model:** Latihan model tersebut pada data yang telah dpersiapkan sebelumnya. Dalam proses ini, akan menggunakan fungsi kehilangan (loss function) dan optimizer untuk menyesuaikan berbagai parameter model.
- **Pemrediksi dan evaluasi**: Setelah model telah latihan, dapat menggunakannya untuk membuat prediksi pada data yang baru. juga dapat mengevaluasi kinerja model dengan menggunakan metrik seperti kemampuan, ketepatan, atau F1-score.

- **Pengiriman model:** Jika ingin menggunakan model ini untuk membuat prediksi pada data baru atau mengoptimalkan model untuk meningkatkan kinerja, dapat menggunakan teknik pengiriman (serialization) untuk menyimpan model ke file dan mengimpornya ke sistem lain.

Dalam mengikuti langkah-langkah ini, PyTorch digunakan untuk mengimplementasikan model deep learning dan mengoptimalkannya untuk meningkatkan kinerja pada tugas yang diberikan. akan belajar cara menggunakan fitur-fitur PyTorch, seperti torch.nn, torch.optim, dan torch.nn.functional, untuk membangun dan mengoptimalkan model deep learning.

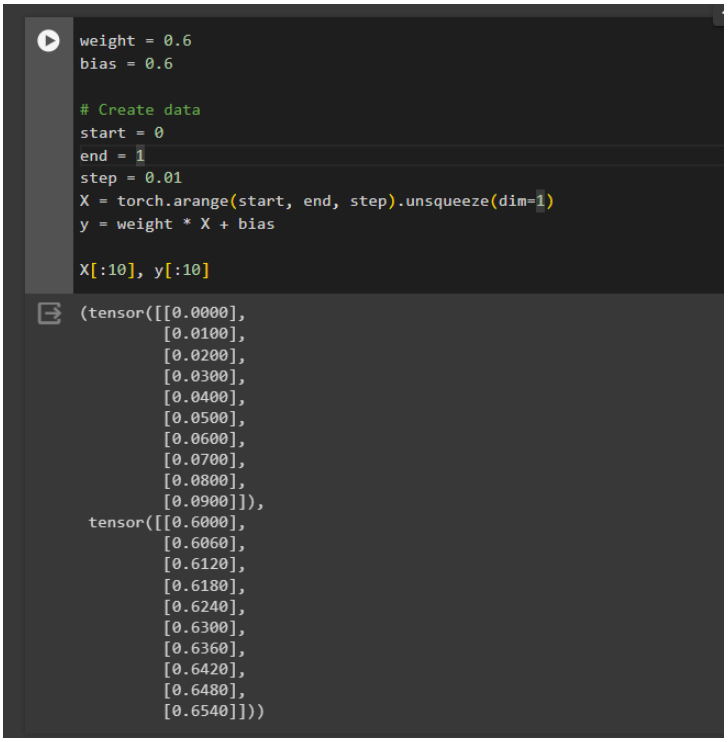


```
[ ]  import torch
     from torch import nn # nn contains all of PyTorch
     import matplotlib.pyplot as plt

     # Check PyTorch version
     torch.__version__

     '2.1.0+cu121'
```

Pada langkah awal seperti biasa dengan pemanggilan beberapa library yang dibutuhkan untuk pemanggilan datasetnya. Selanjutnya yaitu mencetak version dari torch 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]]))
```

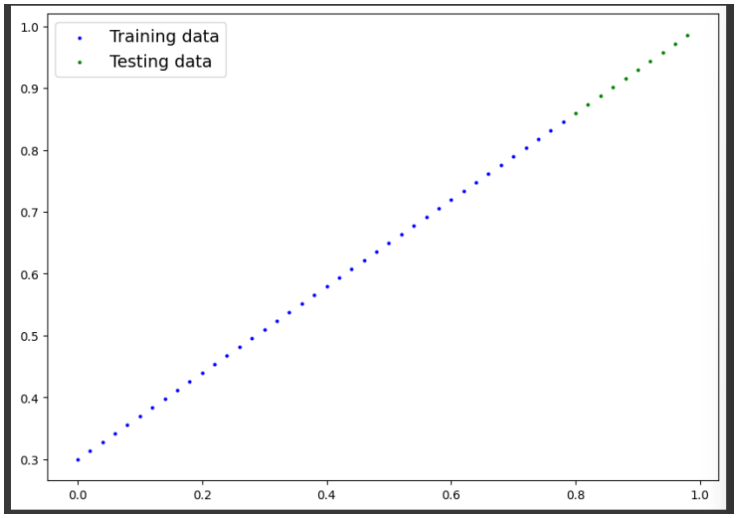
Pada awal line mendefinisikan bobot (weight) dan bias dengan nilai masing-masing 0.6. membuat data input (X) menggunakan fungsi torch.arange(), yang menghasilkan nilai mulai dari 0 hingga 1 dengan langkah sebesar 0.01. Kemudian, dengan menggunakan metode

.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});
```



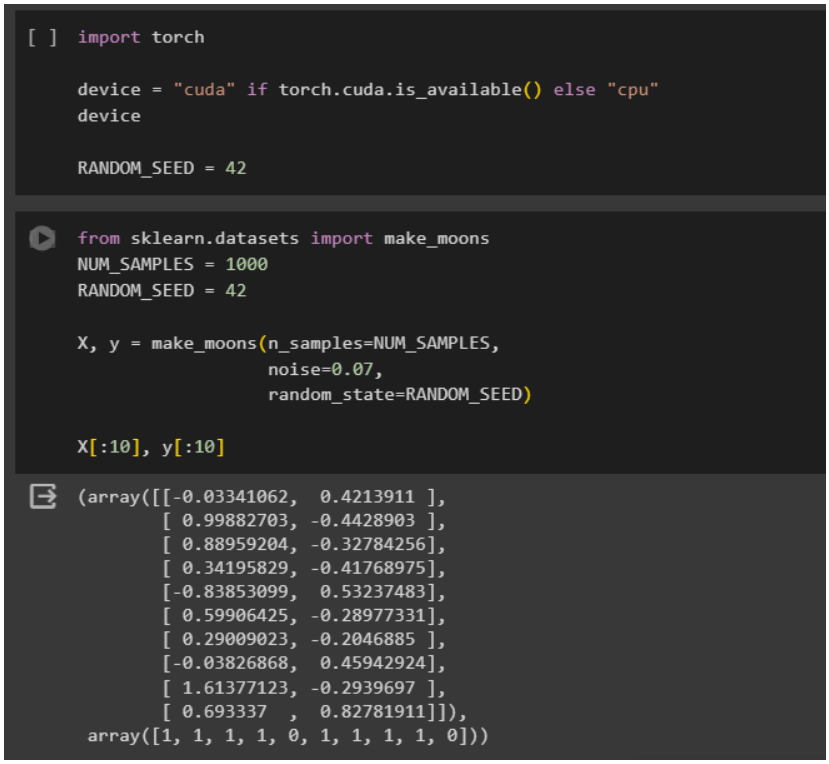
Fungsi plot_predictions digunakan untuk membuat plot yang memvisualisasikan data latih, data uji, dan perbandingan prediksi (jika ada). Berikut adalah penjelasan singkat mengenai fungsinya:

- plt.figure(figsize=(10, 7)): Membuat figur (gambar) dengan ukuran 10x7 inci menggunakan matplotlib.

- plt.scatter(train_data, train_labels, c="b", s=4, label="Training data"): Membuat scatter plot (diagram pencar) 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 bawah ini 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 di visualisasikan.

3. CHAPTER 002 PyTorch Neural Network Classification

PyTorch Neural Network Classification merupakan penggunaan PyTorch untuk membangun dan melatih jaringan saraf dalam konteks klasifikasi. Langkah-langkah umum dalam PyTorch untuk klasifikasi meliputi pembuatan dataset kustom, pembangunan jaringan saraf, pelatihan jaringan saraf, dan evaluasi kinerja model. Proses ini melibatkan penggunaan berbagai modul PyTorch seperti torch.nn untuk membangun model, torch.optim untuk mengatur optimizer, dan fungsi kehilangan (loss function) untuk mengukur kinerja model. Selain itu, langkah-langkah ini juga melibatkan 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 menggunakan fungsi make_moons dari scikit-learn untuk membuat dataset yang terdiri dari dua set data dengan bentuk bulan sabit (moon shapes). from sklearn.datasets import make_moons: Mengimpor fungsi make_moons dari modul datasets di scikit-learn. NUM_SAMPLES = 1000: Menetapkan jumlah sampel (samples) yang ingin dihasilkan, dalam hal ini, 1000. RANDOM_SEED = 42: Menetapkan nilai benih acak untuk memastikan reproduktibilitas hasil, dalam hal ini, 42. X, y = make_moons(n_samples=NUM_SAMPLES, noise=0.07, random_state=RANDOM_SEED): Menggunakan fungsi make_moons untuk membuat dataset. Parameter yang digunakan: n_samples=NUM_SAMPLES: Menentukan jumlah sampel yang ingin dihasilkan, sesuai dengan nilai yang telah ditetapkan sebelumnya. noise=0.07: Menentukan tingkat kebisingan pada data, dalam hal ini, 0.07. random_state=RANDOM_SEED: Menetapkan benih acak untuk memastikan hasil yang konsisten.

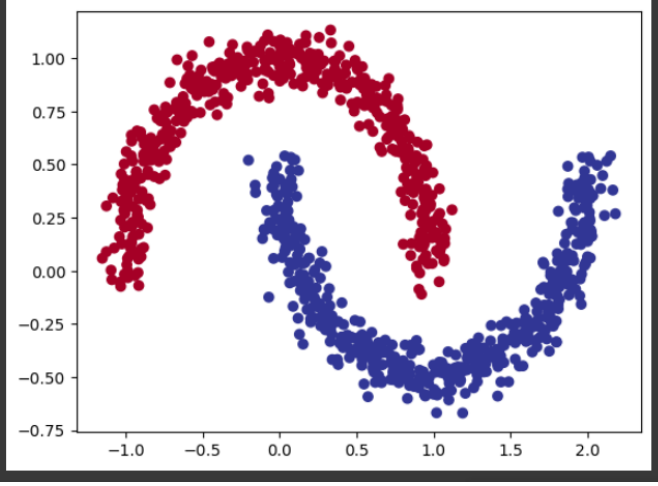
Hasilnya, variabel X akan berisi array dua dimensi yang mewakili koordinat dari setiap sampel, dan variabel y akan berisi array satu dimensi yang berisi label kelas untuk setiap sampel (0 atau 1), sesuai dengan bentuk bulan sabit yang dihasilkan oleh fungsi make_moons. Sebagai contoh, kode ini mencetak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

data_df = pd.DataFrame({"X0": X[:, 0], "X1": X[:, 1], "y": y}): Membuat DataFrame menggunakan ps. 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. data_df.head(): Mencetak lima baris pertama dari DataFrame yang baru dibuat. Metode .head() digunakan untuk memberikan tampilan singkat dari struktur dan isi DataFrame. Dengan kata lain, DataFrame data_df terdiri dari tiga kolom: "X0" dan "X1" yang berisi koordinat dari setiap sampel, dan "y" yang berisi label kelas (0 atau 1) untuk setiap sampel. Ini memberikan cara yang lebih terstruktur untuk menyimpan dan memanipulasi dataset dibandingkan dengan array numpy mentah.

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



import matplotlib.pyplot as plt: Mengimpor library matplotlib dan memberi alias sebagai plt. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu): Membuat scatter plot menggunakan matplotlib. X[:, 0] dan X[:, 1] digunakan sebagai koordinat sumbu x dan y, sedangkan c=y

menentukan warna untuk setiap titik berdasarkan label kelas yang disimpan dalam array y. cmap=plt.cm.RdYlBu menentukan peta warna yang digunakan (dari merah ke kuning hingga biru).

Hasilnya adalah scatter plot di mana setiap titik merepresentasikan satu sampel, warna titik menunjukkan label kelasnya, dan posisi titik menunjukkan koordinat dari setiap sampel pada sumbu x dan y. Ini memberikan visualisasi yang berguna untuk memahami distribusi dan pola dari dataset "moons" yang dihasilkan.

```
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 merupakan contoh implementasi pelatihan dan evaluasi model menggunakan PyTorch. epochs = 1000: Menetapkan jumlah epoch (siklus pelatihan) yang ingin dilakukan, dalam hal ini, 1000 epoch. Melakukan loop melalui epoch:

a. model_0.train(): Menetapkan model ke mode pelatihan. Ini memastikan bahwa parameter model dapat di-update selama proses pelatihan.

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 (dalam hal ini, tampaknya 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 (mengupdate parameter model).

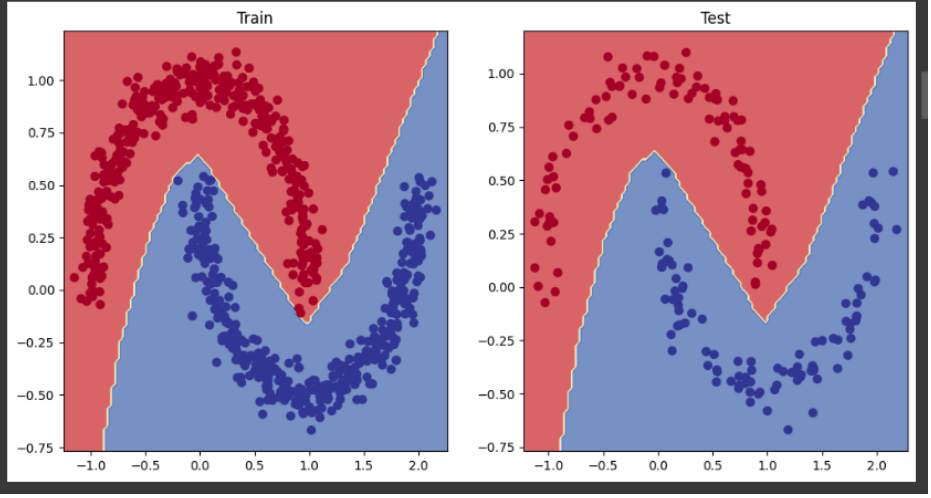
e. model_0.eval(): Menetapkan model ke mode evaluasi. Ini mematikan dropout atau pengaruh lainnya yang mungkin terjadi selama pelatihan.

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.

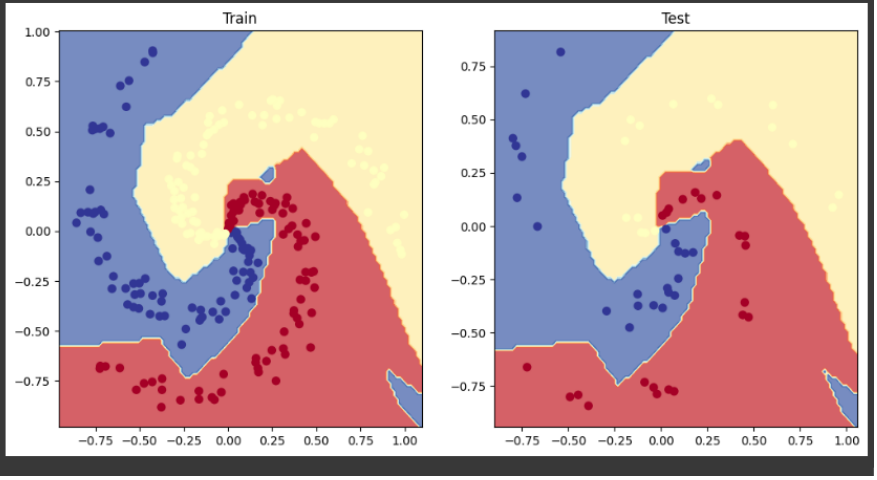
g. Setiap 100 epoch, mencetak hasil pelatihan dan evaluasi, termasuk nilai loss dan akurasi.

```
[ ]  plt.figure(figsize=(12, 6))
     plt.subplot(1, 2, 1)
     plt.title("Train")
     plot_decision_boundary(model_0, X_train, y_train)
     plt.subplot(1, 2, 2)
     plt.title("Test")
     plot_decision_boundary(model_0, X_test, y_test)
```



Dengan menggunakan dua subplot, dapat dengan mudah membandingkan bagaimana model berkinerja pada data latih dan data uji, serta melihat bagaimana model tersebut mengklasifikasikan sampel-sampel pada batas keputusannya. Fungsi plot_decision_boundary digunakan untuk memvisualisasikan 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, dapat membandingkan bagaimana model_1 berkinerja pada data latih dan data uji, serta melihat bagaimana model tersebut mengklasifikasikan sampel-sampel pada batas keputusannya. Fungsi plot_decision_boundary

digunakan untuk memvisualisasikan batas keputusan model pada data dua dimensi, dan mungkin berisi logika khusus tergantung pada implementasinya di dalam kode.