Nama :Juan Anemao Sokhi Zidomi

NIM :1103204007

Kelas :TK-44-G6

PePytorch Neural Network Classification Penjelasan langkah-langkah dan source code

1. Program

```
from sklearn.datasets import make_circles

n_samples = 1000

X, y = make_circles(n_samples,

                    noise=0.03,

                    random_state=42)
```

Analisa:Program diatas melakukan pembuatan dataset dengan menggunakan import library make_circles dengan jumlah sample=100, noise=0.03 ,dilakukan secara random

2. Program

```
print(f"First 5 X features:\n{X[:5]}")

print(f"\nFirst 5 y labels:\n{y[:5]}")
```

Hasil

```
First 5 X features:
[[ 0.75424625  0.23148074]
 [-0.75615888  0.15325888]
 [-0.81539193  0.17328203]
 [-0.39373073  0.69288277]
 [ 0.44220765 -0.89672343]]

First 5 y labels:
[1 1 1 1 0]
```

Analisa:Program diata menampilkan 5 baris pertama dari variabel X dan variabel y

3. Program

```
import pandas as pd

circles = pd.DataFrame({"X1": X[:, 0],

    "X2": X[:, 1],

    "label": y

})

circles.head(10)
```

Output:Program diatas melakukan pengubahan dataset menjadi dataframe dengan menggunakan library pandas ,kemudian melakukan pembuatan kolom X1 dan X2 dengan mengambil nilai dari variabel X ,kemudian mengambil nilai y dimasukkan ke kolom label kemudian menampilkan 10 baris pertama

4. Program

```
circles.label.value_counts()
```

Hasil

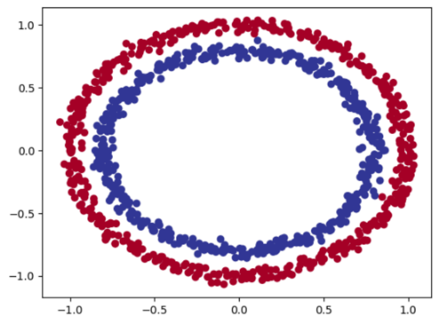
```
1    500
0    500
Name: label, dtype: int64
```

Analisa:Program diatas melakukan perhitungan jumlah nilai label yaitu 0 dan 1 masing-masing sebesar 500

5. Program

```
import matplotlib.pyplot as plt

plt.scatter(x=X[:, 0],

            y=X[:, 1],

            c=y,

            cmap=plt.cm.RdYlBu);
```

Hasil



Analisa:Program diatas melakukan visualisasi menggunakan matplotlib untuk memvisualisasikan nilai X dan y

6. Program

```
X.shape, y.shape
```

Hasil

```
((1000, 2), (1000,))
```

Analisa:Program diatas menampilkan bentuk dari variabel X dan y ,X dan y memiliki 1000 data . X memiliki 2 dimensi sedangkan y memiliki 1 dimensi

7. Program

```
X_sample = X[0]

y_sample = y[0]

print(f"Values for one sample of X: {X_sample} and the same for y:
{y_sample}")

print(f"Shapes for one sample of X: {X_sample.shape} and the same for y:
{y_sample.shape}")
```

Hasil

```
Values for one sample of X: [0.75424625 0.23148074] and the same for y: 1
Shapes for one sample of X: (2,) and the same for y: ()
```

Analisa:Program tersebut mengambil nilai X dan y dari indeks 1 kemudian menampilkannya ,dan menampilkan bentuk data

8. Program

```
import numpy as np

import torch

X=np.array(X)

y=np.array(y)

X = torch.from_numpy(X).type(torch.float)

y = torch.from_numpy(y).type(torch.float)

X[:5], y[:5]
```

Hasil

```
(tensor([[ 0.7542,  0.2315],
         [-0.7562,  0.1533],
         [-0.8154,  0.1733],
         [-0.3937,  0.6929],
         [ 0.4422, -0.8967]]),
 tensor([1., 1., 1., 1., 0.]))
```

Analisa:Program diatas melakukan pengubahan nilai ke array numpy dan menjadi tipe data torch float

9. Program

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_sta
te=42)

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

Hasil

```
(800, 200, 800, 200)
```

Analisa:Program diatas melakukan import library train_test_split kemudian melakukan pembagian dataset menjadi 2 bagian train dan test dengan test_size=0.2 kemudian menampilkan jumlah data dari X dan y setelah di lakukan pemabagian data train dan data test

10. Program

```
import torch

from torch import nn

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

device
```

Hasil

```
'cpu'
```

Analisis:Program diatas melakukan import torch dan nn kemudian melakukan pemindaian device dan terdeteksi yaitu CPi

11. Program

```
class CircleModelV0(nn.Module):

  def __init__(self):

    super().__init__()

    self.layer_1=nn.Linear(in_features=2,out_features=5)

    self.layer_2=nn.Linear(in_features=5,out_features=1)

  def forward(self, x):

    return self.layer_2(self.layer_1(x))

model_0 = CircleModelV0().to(device)

model_0
```

Hasil

```
CircleModelV0(
  (layer_1): Linear(in_features=2, out_features=5, bias=True)
  (layer_2): Linear(in_features=2, out_features=5, bias=True)
)
```

Analisa:Program diatas melakukan inisiasi model NN dengan menggunakan kelas kemudian terdapat mendefenisikan layer pertama dari model dengan 2 fitur masukan dan 5 fitur keluaran dan layer 2 dengan 5 fitur masukan dan 1 fitur keluaran sebagai layer connetcted (Linear)

12. Program

```
model_0=nn.Sequential(

    nn.Linear(in_features=2,out_features=5),

    nn.Linear(in_features=5,out_features=1)

).to(device)

model_0
```

Hasil

```
Sequential(
  (0): Linear(in_features=2, out_features=5, bias=True)
  (1): Linear(in_features=5, out_features=1, bias=True)
)
```

Analisa:Program diatas melakukan Sequential dengan layer pertama dan layer ke 2 yang sebagai layer fully connected Dimana layer 2 sebagai layer output yang menghasilkan prediksi

13. Program

```
untrained_preds=model_0(X_test.to(device))

print(f"Length of
Predictions:{len(untrained_preds)},Shape:{untrained_preds.shape}")

print(f"Length of test samples:{len(y_test)},Shape:{y_test.shape}")

print(f"\nFirst 10 predictions:\n{untrained_preds[:10]}")
```

Hasil

```
Length of Predictions:200,Shape:torch.Size([200, 1])
Length of test samples:200,Shape:torch.Size([200])

First 10 predictions:
tensor([[-0.3030],
        [-0.2402],
        [-0.4600],
        [-0.2585],
        [-0.4478],
        [-0.4096],
        [-0.2424],
        [-0.2820],
        [-0.4603],
        [-0.2358]], grad_fn=<SliceBackward0>)

First 10 test labels:
tensor([1., 0., 1., 0., 1., 1., 0., 0., 1., 0.])
```

Analisa: Program diatas menampilkan 10 baris pertama dari prediksi dan label ,dengan jumlah prediksi dan bentuk dari data

14. Program

```
loss_fn=nn.BCEWithLogitsLoss()

optimizer=torch.optim.SGD(params=model_0.parameters(),lr=0.1)

def accuracy_fn(y_true,y_pred):

  correct=torch.eq(y_true,y_pred).sum().item()

  acc=(correct/len(y_pred))*100

  return acc

y_logits=model_0(X_test.to(device))[:5]

y_logits
```

Hasil

```
tensor([[-0.3030],
        [-0.2402],
        [-0.4600],
        [-0.2585],
        [-0.4478]], grad_fn=<SliceBackward0>)
```

Analisa:Program diatas melakukan pembuatan fungsi untuk menghitung loss, menghitung akurasi kemudian melakukan forward pass

15. Program

```
y_pred_probs=torch.sigmoid(y_logits)

y_pred_probs
```

Hasil

```
tensor([[0.4248],
        [0.4402],
        [0.3870],
        [0.4357],
        [0.3899]], grad_fn=<SigmoidBackward0>)
```

Analisa:Program diatas menggunakan fungsi sigmoid kemudian menampilkan hasil prediksi dari fungsi sigmoid

16. Program

```
y_preds=torch.round(y_pred_probs)

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

print(torch.eq(y_preds.squeeze(),y_pred_labels.squeeze()))

y_preds.squeeze()
```

Hasil

```
tensor([True, True, True, True, True])
tensor([0., 0., 0., 0., 0.], grad_fn=<SqueezeBackward0>)
```

Analisa:Program diatas melakukan prediksi biner dari probabilitas yang dibuat model dan membandingkannya dengan prediksi biner lainnya

17. Program

```
y_test[:5]
```

Hasil

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

Analisa:Program diatas menampilkan 5 baris y_test untuk melakukan perbandingan hasil prediksi dari model yang telah dibuat dengan nilai aslinya

18. Program

```
torch.manual_seed(42)

epochs = 100

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 = torch.round(torch.sigmoid(y_logits))

    loss = loss_fn(y_logits, y_train)

    acc = accuracy_fn(y_true=y_train, y_pred=y_pred)

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()
```

```
 model_0.eval()

    with torch.no_grad():  # Use torch.no_grad() instead of
torch.inference_model()

        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 = accuracy_fn(y_true=y_test, y_pred=test_pred)

    if epoch % 10 == 0:

        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy:
{acc:.2f}% | Total Loss: {test_loss} | Total Acc: {test_acc:.2f}")
```

Hasil

```
Epoch: 0 | Loss: 0.71081, Accuracy: 50.00% | Total Loss: 0.7126001119613647 | Total Acc: 50.00
Epoch: 10 | Loss: 0.69968, Accuracy: 50.00% | Total Loss: 0.7027788758277893 | Total Acc: 50.00
Epoch: 20 | Loss: 0.69564, Accuracy: 44.62% | Total Loss: 0.6992220878601074 | Total Acc: 42.00
Epoch: 30 | Loss: 0.69415, Accuracy: 46.75% | Total Loss: 0.6978858113288879 | Total Acc: 46.00
Epoch: 40 | Loss: 0.69360, Accuracy: 49.25% | Total Loss: 0.6973321437835693 | Total Acc: 46.00
Epoch: 50 | Loss: 0.69337, Accuracy: 50.38% | Total Loss: 0.6970524787902832 | Total Acc: 46.00
Epoch: 60 | Loss: 0.69328, Accuracy: 50.50% | Total Loss: 0.6968697309494019 | Total Acc: 44.50
Epoch: 70 | Loss: 0.69322, Accuracy: 50.50% | Total Loss: 0.6967236399650574 | Total Acc: 46.00
Epoch: 80 | Loss: 0.69319, Accuracy: 51.00% | Total Loss: 0.6965939998626709 | Total Acc: 47.00
Epoch: 90 | Loss: 0.69316, Accuracy: 50.88% | Total Loss: 0.6964743733406067 | Total Acc: 47.00
```

Analisa:Program diatas melakukan pelatihan model dengan jumlah epoch=100 ,dengan menampilkan jumlah accuracy dan loss jika setiap %10 ==0

19. Program

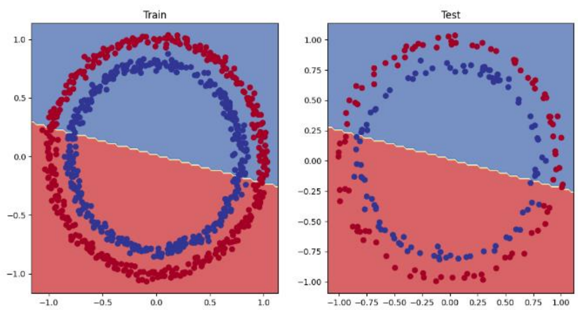
```
import requests

from pathlib import Path

if Path("helper_functions.py").is_file():

  print("helper_functions.py already exists, skipping download")

else :

  print("Downloading helper_functions.py")

  request=requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-
deep-learning/main/helper_functions.py")

  with open("helper_functions.py","wb") as f:

    f.write(request.content)

from helper functions import plot predictions, plot decision boundary
```

Analisa:Program diatas melakukan pemeriksaan direktori jika belum ada maka akan mendownload modul request sebagai helper_functions.py

20. Program

```
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)
```

Hasil



Analisa:Program diatas melakukan visualisasi hasil train dan hasil test dengan menggunakan library matplotlib

21. Program

```
weight=0.7

bias=0.3

start=0

end=1

step=0.01

X_regression=torch.arange(start,end,step).unsqueeze(dim=1)

y_regression=weight *X_regression + bias

print(len(X_regression))

X_regression[:5], y_regression[:5]
```

Hasil

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

Analisa:Program diatas melakukan proses regresi di pytorch kemudian menampilkan hasilnya

22. Program

```
train_split = int(0.8 * len(X_regression)) # 80% of data used for training set

X_train_regression, y_train_regression = X_regression[:train_split], y_regression[:train_split]

X_test_regression, y_test_regression = X_regression[train_split:], y_regression[train_split:]

print(len(X_train_regression),

    len(y_train_regression),

    len(X_test_regression),

    len(y_test_regression))
```

Hasil

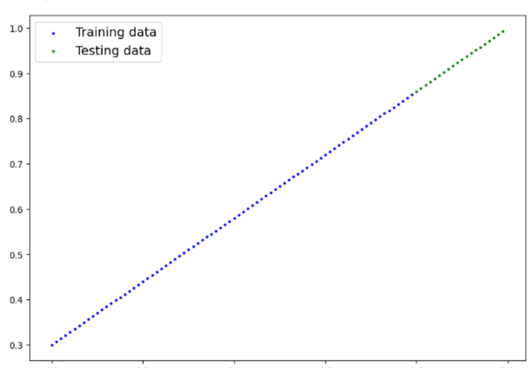
```
80 80 20 20
```

Analisa:Program diatas melakukan pemisahan dataset yaitu Data train 80% dan Data Test 20 % ,kemudian melakukan split pada data X_train_regression dan y_train_regression ,sama juga halnya pada X_test_regression dan y_test_regression ,kemudian menampilkan jumlah data pada setiap variabel

23. Program

```
plot_predictions(train_data=X_train_regression,

    train_labels=y_train_regression,

    test_data=X_test_regression,

    test_labels=y_test_regression

);
```

Hasil



Analisa:Program diatas melakukan visualisasi hasil prediksi dan train dengan menggunakan library matplotlib

24. Program

```
class CircleModelV1(nn.Module):

    def __init__(self):

        super().__init__()

        self.layer_1 = nn.Linear(in_features=2, out_features=10)

        self.layer_2 = nn.Linear(in_features=10, out_features=10)

        self.layer_3 = nn.Linear(in_features=10, out_features=1)

    def forward(self, x):

        return self.layer_3(self.layer_2(self.layer_1(x)))

model_1 = CircleModelV1().to(device)

model_1
```

Hasil

```
Sequential(
  (0): Linear(in_features=1, out_features=10, bias=True)
  (1): Linear(in_features=10, out_features=10, bias=True)
  (2): Linear(in_features=10, out_features=1, bias=True)
)
```

Analisa:Program diatas melakukan pembuatan model dengan 3 layer

25. Program

```
loss_fn=nn.BCEWithLogitsLoss()

optimizer=torch.optim.SGD(model_1.parameters(),lr=0.1)

torch.manual_seed(42)

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):

    y_logits = model_1(X_train).squeeze()

    y_pred = torch.round(torch.sigmoid(y_logits)) # logits -> predicition
probabilities -> prediction labels

    loss = loss_fn(y_logits, y_train)

    acc = accuracy_fn(y_true=y_train,

                      y_pred=y_pred)

    optimizer.zero_grad()
```

```
  model_1.eval()

    with torch.inference_mode():

        test_logits = model_1(X_test).squeeze()

        test_pred = torch.round(torch.sigmoid(test_logits))

        test_loss = loss_fn(test_logits,

                        y_test)

        test_acc = accuracy_fn(y_true=y_test,

                            y_pred=test_pred)

    if epoch % 100 == 0:

        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% |
Test loss: {test_loss:.5f}, Test acc: {test_acc:.2f}%")
```

Hasil

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

Analisa:Program diatas melakukan pelatihan model dengan jumlah epoch =1000 dengan ,menampilkan nilai akurasi dan loss jika epoch dibagi 100 ==0

26. Program

```
model_2 = nn.Sequential(

    nn.Linear(in_features=1, out_features=10),

    nn.Linear(in_features=10, out_features=10),

    nn.Linear(in_features=10, out_features=1)

).to(device)

model_2
```

Hasil

```
Sequential(
  (0): Linear(in_features=1, out_features=10, bias=True)
  (1): Linear(in_features=10, out_features=10, bias=True)
  (2): Linear(in_features=10, out_features=1, bias=True)
)
```

Analisa:Program diatas melakukan pembuatan arstiktur dengan layer pertama memiliki 1 feature input dan feature output 10 ,layer 2 memiliki 10 input feature dan output feature, dan layer ketiga menjadi 10 input feature dan 1 output feature

27. Program

```
loss_fn = nn.L1Loss()

optimizer = torch.optim.SGD(model_2.parameters(), lr=0.1)

torch.manual_seed(42)

epochs = 1000

X_train_regression, y_train_regression = X_train_regression.to(device), y_train_regression.to(device)

X_test_regression, y_test_regression = X_test_regression.to(device), y_test_regression.to(device)

for epoch in range(epochs):

    y_pred = model_2(X_train_regression)

    loss = loss_fn(y_pred, y_train_regression)

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    model_2.eval()

    with torch.inference_mode():

      test_pred = model_2(X_test_regression)

      test_loss = loss_fn(test_pred, y_test_regression)

    if epoch % 100 == 0:

        print(f"Epoch: {epoch} | Train loss: {loss:.5f}, Test loss: {test_loss:.5f}")
```

Hasil

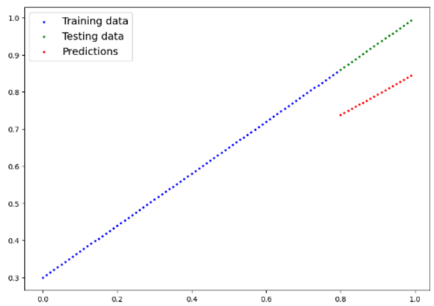
```
Epoch: 0 | Train loss: 0.75986, Test loss: 0.54143
Epoch: 100 | Train loss: 0.09309, Test loss: 0.02901
Epoch: 200 | Train loss: 0.07376, Test loss: 0.02850
Epoch: 300 | Train loss: 0.06745, Test loss: 0.00615
Epoch: 400 | Train loss: 0.06107, Test loss: 0.02004
Epoch: 500 | Train loss: 0.05698, Test loss: 0.01061
Epoch: 600 | Train loss: 0.04857, Test loss: 0.01326
Epoch: 700 | Train loss: 0.06109, Test loss: 0.02127
Epoch: 800 | Train loss: 0.05600, Test loss: 0.01425
Epoch: 900 | Train loss: 0.05571, Test loss: 0.00603
```

Analisa:Program diatas melakukan pelatihan model pada model_2 dengan menampilkan nilai akurasi dan loss jika setiap epoch dibagi 100 =0 dengan jumlah epoch =1000

28. Program

```
model_2.eval()
with torch.inference_mode():
    y_preds = model_2(X_test_regression)
plot_predictions(train_data=X_train_regression.cpu(),
                 train_labels=y_train_regression.cpu(),
                 test_data=X_test_regression.cpu(),
                 test_labels=y_test_regression.cpu(),
                 predictions=y_preds.cpu());
```

Hasil

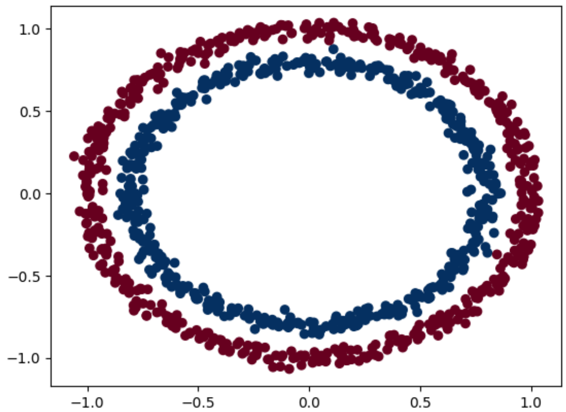


Analisa:Program diatas melakukan visualisasi hasil train ,testing dan hasil prediksi dari model 2

29. Program

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
n_samples = 1000
X, y = make_circles(n_samples=1000,
    noise=0.03,
    random_state=42,
)
```

Hasil



Analisa:Ptogram diatas melakukan pembuatan data dengan jumlah sample =1000 secara random denngan berbentuk circles kemudian menampilkan hasil visualisasi dengan menggunakan matplotlib

30. Program

```
import torch
from sklearn.model_selection import train_test_split
X = torch.from_numpy(X).type(torch.float)
y = torch.from_numpy(y).type(torch.float)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42
)
X train[:5], y train[:5]
```

Hasil

```
(tensor([[ 0.6579, -0.4651],
         [ 0.6319, -0.7347],
         [-1.0086, -0.1240],
         [-0.9666, -0.2256],
         [-0.1666,  0.7994]]),
 tensor([1., 0., 0., 0., 1.]))
```

Analisa:Program diatas melakukan pemisahan data train dan data test dan melakukan dataset menjadi array numpy dengan bertipe float .Proporsi data train data test yaitu 20 % data test kemudian menampilkan baris pada setiap variabel

31. Program

```
from torch import nn

class CircleModelV2(nn.Module):

    def __init__(self):

        super().__init__()

        self.layer_1 = nn.Linear(in_features=2, out_features=10)

        self.layer_2 = nn.Linear(in_features=10, out_features=10)

        self.layer_3 = nn.Linear(in_features=10, out_features=1)

        self.relu = nn.ReLU()

    def forward(self, x):

       return
self.layer_3(self.relu(self.layer_2(self.relu(self.layer_1(x)))))

model_3 = CircleModelV2().to(device)

print(model_3)
```

Hasil

```
CircleModelV2(
  (layer_1): Linear(in_features=2, out_features=10, bias=True)
  (layer_2): Linear(in_features=10, out_features=10, bias=True)
  (layer_3): Linear(in_features=10, out_features=1, bias=True)
  (relu): ReLU()
)
```

Analisa:Program diatas membuat arstitektur dengan 3 layer fully connted dengan jumlah input_featute dan output feature yang berbeda dan menambahkan 1 layer relu

32. Program

```
loss_fn = nn.BCEWithLogitsLoss()

optimizer = torch.optim.SGD(model_3.parameters(), lr=0.1)

torch.manual_seed(42)

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):

        y_logits = model_3(X_train).squeeze()

    y_pred = torch.round(torch.sigmoid(y_logits)) # logits -> prediction
probabilities -> prediction labels

    loss = loss_fn(y_logits, y_train) # BCEWithLogitsLoss calculates loss
using logits

    acc = accuracy_fn(y_true=y_train,

                      y_pred=y_pred)
```

```
optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    model_3.eval()igmoid(test_logits)) # logits -> prediction probabilities
-> prediction labels

      test_loss = loss_fn(test_logits, y_test)

      test_acc = accuracy_fn(y_true=y_test,

                             y_pred=test_pred)

    if epoch % 100 == 0:

        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% |
Test Loss: {test_loss:.5f}, Test Accuracy: {test_acc:.2f}%")
```

Hasil

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

Analisa:Program diatas melakukan pelatihan model 2 dengan menampilkan akurasi dan loss jika jumlah epochnya habis dibagi 100 maka akan menampilkan akurasi dan loss

33. Program

```
model_3.eval()

with torch.inference_mode():

    y_preds = torch.round(torch.sigmoid(model_3(X_test))).squeeze()

y_preds[:10], y[:10] # want preds in same format as truth labels
```

Hasil

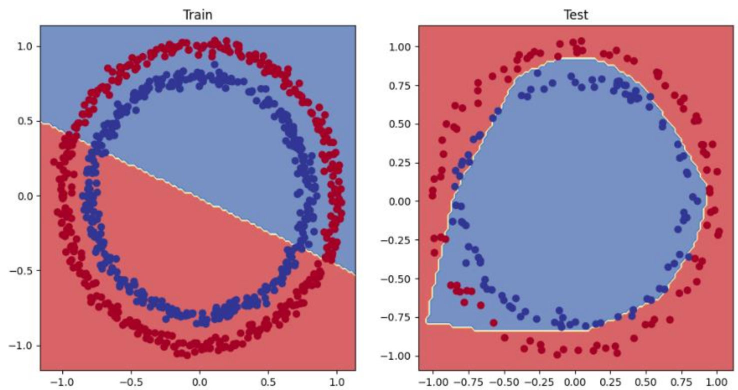
```
(tensor([1., 0., 1., 0., 0., 1., 0., 0., 1., 0.]),
 tensor([1., 1., 1., 1., 0., 1., 1., 1., 1., 0.]))
```

Analisa:Program diatas melakukan evaluasi model

34. Program

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.title("Train")

plot_decision_boundary(model_1, X_train, y_train) # model_1 = no non-
linearity

plt.subplot(1, 2, 2)

plt.title("Test")

plot_decision_boundary(model_3, X_test, y_test) # model_3 = has non-
linearity
```

Hasil



Analisa:Program diatas melakukan visualisasi pengujian

35. Program

```
A = torch.arange(-10, 10, 1, dtype=torch.float32)

A
```

Hasil

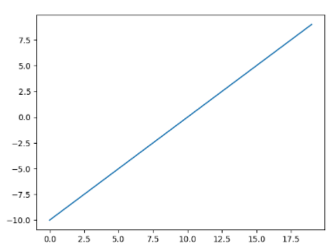
```
tensor([-10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,  -3.,  -2.,  -1.,   0.,   1.,
          2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.])
```

Analisa:Program diatas melakukan pembuatan tensor array dengan nilai -10 sebagai nilai awal ,10 nilai akhir rentang dan 1 adalah jarak rentang

36. Program

```
plt.plot(A);
```

Hasil



Analisa:Program diatas melakukan visualisasi hasil pembuatan tensor array

37. Program

```
def sigmoid(x):

  return 1 / (1 + torch.exp(-x))

sigmoid(A)
```

Hasil

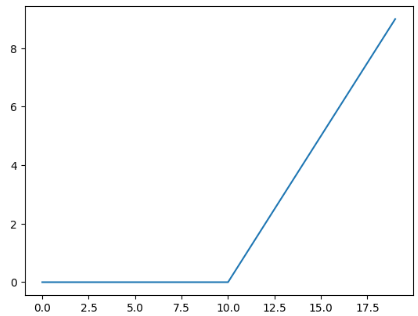
```
tensor([4.5398e-05, 1.2339e-04, 3.3535e-04, 9.1105e-04, 2.4726e-03, 6.6929e-03,
        1.7986e-02, 4.7426e-02, 1.1920e-01, 2.6894e-01, 5.0000e-01, 7.3106e-01,
        8.8080e-01, 9.5257e-01, 9.8201e-01, 9.9331e-01, 9.9753e-01, 9.9909e-01,
        9.9966e-01, 9.9988e-01])
```

Analisa:Program diatas melakukan pembuatan fungsi sigmoid

38. Program

```
plt.plot(sigmoid(A));
```

Hasil



Analisa:Program diatas melakukan visualisasi hasil dari pembuatan fungsi sigmoid

39. Program

```
def sigmoid(x):

  return 1 / (1 + torch.exp(-x))

sigmoid(A)
```

Hasil

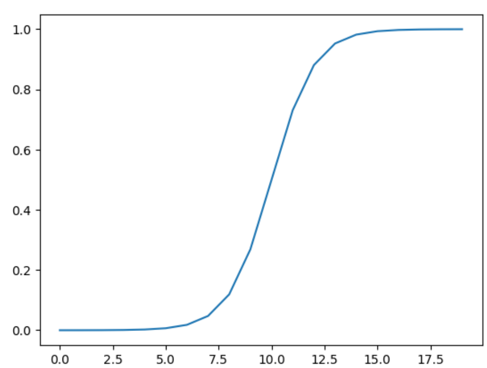
```
tensor([4.5398e-05, 1.2339e-04, 3.3535e-04, 9.1105e-04, 2.4726e-03, 6.6929e-03,
        1.7986e-02, 4.7426e-02, 1.1920e-01, 2.6894e-01, 5.0000e-01, 7.3106e-01,
        8.8080e-01, 9.5257e-01, 9.8201e-01, 9.9331e-01, 9.9753e-01, 9.9909e-01,
        9.9966e-01, 9.9988e-01])
```

Analisa:Program tersebut melakukan pembuatan fungsi sigmoid kemudian menampilkan hasil sigmoid

40. Program

```
plt.plot(sigmoid(A));
```

Hasil

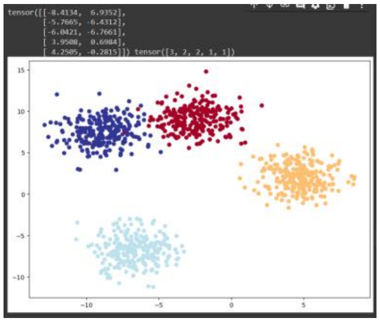


Analisa:Program diatas melakukan visualisasi hasil fungsi sigmoid

41. Program

```
import torch

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.model_selection import train_test_split

NUM_CLASSES = 4

NUM_FEATURES = 2

RANDOM_SEED = 42


# 1. Create multi-class data

X_blob, y_blob = make_blobs(n_samples=1000,

    n_features=NUM_FEATURES, # X features

    centers=NUM_CLASSES, # y labels

    cluster_std=1.5, # give the clusters a little shake up (try changing
this to 1.0, the default)

    random_state=RANDOM_SEED

)

X_blob = torch.from_numpy(X_blob).type(torch.float)

y_blob = torch.from_numpy(y_blob).type(torch.LongTensor)

print(X_blob[:5], y_blob[:5])

X_blob_train, X_blob_test, y_blob_train, y_blob_test =
train_test_split(X_blob,

    y_blob,

    test_size=0.2,

    random_state=RANDOM_SEED

)

plt.figure(figsize=(10, 7))

plt.scatter(X_blob[:, 0], X_blob[:, 1], c=y_blob, cmap=plt.cm.RdYlBu);
```

Hasil



Analisa:Program diatas melakukan clusterisasi

42. Program

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

device
```

Hasil

```
'cpu'
```

Analisa:Program diatas melakukan pencarian resource berupa cuda ,jika tidak memiliki cuda maka dapat diganti dengan CPU

43. Program

```
from torch import nn


# Build model

class BlobModel(nn.Module):

    def __init__(self, input_features, output_features, hidden_units=8):


        super().__init__()

        self.linear_layer_stack = nn.Sequential(

            nn.Linear(in_features=input_features, out_features=hidden_units),

            nn.Linear(in_features=hidden_units, out_features=hidden_units),

            nn.Linear(in_features=hidden_units, out_features=output_features), # how many classes are there?

        )

    def forward(self, x):

        return self.linear_layer_stack(x)

model_4 = BlobModel(input_features=NUM_FEATURES,

                    output_features=NUM_CLASSES,

                    hidden_units=8).to(device)

model_4
```

Hasil

```
BlobModel(
  (linear_layer_stack): Sequential(
    (0): Linear(in_features=2, out_features=8, bias=True)
    (1): Linear(in_features=8, out_features=8, bias=True)
    (2): Linear(in_features=8, out_features=4, bias=True)
  )
)
```

Analisa:Program diatas melakukan pembuatan arsitektur model

44. Program

```
loss_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model_4.parameters(),

                             lr=0.1) # exercise: try changing the

model_4(X_blob_train.to(device))[:5]
```

Hasil

```
tensor([[-1.2711, -0.6494, -1.4740, -0.7044],
        [ 0.2210, -1.5439,  0.0420,  1.1531],
        [ 2.8698,  0.9143,  3.3169,  1.4027],
        [ 1.9576,  0.3125,  2.2244,  1.1324],
        [ 0.5458, -1.2381,  0.4441,  1.1804]], grad_fn=<SliceBackward0>)
```

Analisa:Program diatas melakukan pembuatan loss dan optimizer dan kemudian menampilkan 5 baris pertama

45. Program

```
model_4(X_blob_train.to(device))[0].shape, NUM_CLASSES

y_logits = model_4(X_blob_test.to(device))

y_pred_probs = torch.softmax(y_logits, dim=1)

print(y_logits[:5])

print(y_pred_probs[:5])
```

Hasil

```
(torch.Size([4]), 4)
```

```
tensor([[-1.2549, -0.8112, -1.4795, -0.5696],
        [ 1.7168, -1.2270,  1.7367,  2.1010],
        [ 2.2400,  0.7714,  2.6020,  1.0107],
        [-0.7993, -0.3723, -0.9138, -0.5388],
        [-0.4332, -1.6117, -0.6891,  0.6852]], grad_fn=<SliceBackward0>)
tensor([[0.1872, 0.2918, 0.1495, 0.3715],
        [0.2824, 0.0149, 0.2881, 0.4147],
        [0.3380, 0.0778, 0.4854, 0.0989],
        [0.2118, 0.3246, 0.1889, 0.2748],
        [0.1945, 0.0598, 0.1506, 0.5951]], grad_fn=<SliceBackward0>)
```

Analisa:Program diatas melakukan pengujian model 4

46. Program

```
print(y_pred_probs[0])

print(torch.argmax(y_pred_probs[0]))
```

Hasil

```
tensor([0.1872, 0.2918, 0.1495, 0.3715], grad_fn=<SelectBackward0>)
tensor(3)
```

Analisa:Program diatas menampilkan probabilitas hasil pengujian

47. Program

```
torch.manual_seed(42)

epochs = 100

X_blob_train, y_blob_train = X_blob_train.to(device), y_blob_train.to(device)

X_blob_test, y_blob_test = X_blob_test.to(device), y_blob_test.to(device)

for epoch in range(epochs):

    model_4.train()

    y_logits = model_4(X_blob_train) # model outputs raw logits

    y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1) # go from logits -> 
prediction probabilities -> prediction labels

    loss = loss_fn(y_logits, y_blob_train)

    acc = accuracy_fn(y_true=y_blob_train,

                      y_pred=y_pred)

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    model_4.eval()

    with torch.inference_mode():

      test_logits = model_4(X_blob_test)

      test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)

      test_loss = loss_fn(test_logits, y_blob_test)

      test_acc = accuracy_fn(y_true=y_blob_test,

                             y_pred=test_pred)

    if epoch % 10 == 0:

        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Acc: {acc:.2f}% | Test Loss: 
{test_loss:.5f}, Test Acc: {test_acc:.2f}%")
```

Hasil

```
Epoch: 0 | Loss: 1.04324, Acc: 65.50% | Test Loss: 0.57861, Test Acc: 95.50%
Epoch: 10 | Loss: 0.14398, Acc: 99.12% | Test Loss: 0.13037, Test Acc: 99.00%
Epoch: 20 | Loss: 0.08062, Acc: 99.12% | Test Loss: 0.07216, Test Acc: 99.50%
Epoch: 30 | Loss: 0.05924, Acc: 99.12% | Test Loss: 0.05133, Test Acc: 99.50%
Epoch: 40 | Loss: 0.04892, Acc: 99.00% | Test Loss: 0.04098, Test Acc: 99.50%
Epoch: 50 | Loss: 0.04295, Acc: 99.00% | Test Loss: 0.03486, Test Acc: 99.50%
Epoch: 60 | Loss: 0.03910, Acc: 99.00% | Test Loss: 0.03083, Test Acc: 99.50%
Epoch: 70 | Loss: 0.03643, Acc: 99.00% | Test Loss: 0.02799, Test Acc: 99.50%
Epoch: 80 | Loss: 0.03448, Acc: 99.00% | Test Loss: 0.02587, Test Acc: 99.50%
Epoch: 90 | Loss: 0.03300, Acc: 99.12% | Test Loss: 0.02423, Test Acc: 99.50%
```

Analisa:Program diatas melakukan pelatihan model dan menampilkan hasil akurasi dan loss dengan jumlah epoch 100

48. Program

```
model_4.eval()

with torch.inference_mode():

    y_logits = model_4(X_blob_test)
```

Hasil

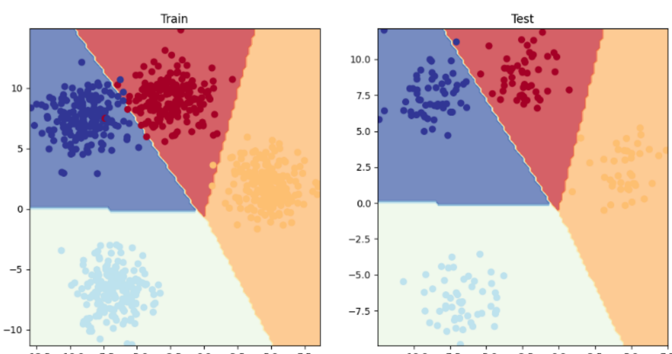
```
tensor([[  4.3377,  10.3539, -14.8948,  -9.7642],
        [  5.0142, -12.0371,   3.3860,  10.6699],
        [ -5.5885, -13.3448,  20.9894,  12.7711],
        [  1.8400,   7.5599,  -8.6016,  -6.9942],
        [  8.0727,   3.2906, -14.5998,  -3.6186],
        [  5.5844, -14.9521,   5.0168,  13.2891],
        [ -5.9739, -10.1913,  18.8655,   9.9179],
        [  7.0755,  -0.7601,  -9.5531,   0.1736],
        [ -5.5918, -18.5990,  25.5310,  17.5799],
        [  7.3142,   0.7197, -11.2017,  -1.2011]])
```

Analisa:Program diata melakukan pengaturan pada evaluasi model dan kemudian menggunakan mode inferensi untuk mendapatkan logits

49. Program

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.title("Train")

plot_decision_boundary(model_4, X_blob_train, y_blob_train)

plt.subplot(1, 2, 2)

plt.title("Test")

plot_decision_boundary(model_4, X_blob_test, y_blob_test)
```

Hasil



Analisa; Program diatas melakukan visualisasi hasil train dan testing