Nama : Hurin Salimah
NIM : 1103200021

Input
```
# Check for GPU
!nvidia-smi
```
Output
```
Thu Jan  4 00:38:06 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   49C    P8               9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Input
```
# Import torch
import torch

# Setup device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```
Output
```
'cuda'
```

1. Make a binary classification dataset with Scikit-Learn's make_moons() function.

Input

```
#membuat dataset sintetis yang terdiri dari dua bulan yang terpisah
from sklearn.datasets import make_moons

#menentukan jumlah sampel dalam dataset
NUM_SAMPLES = 1000
RANDOM_SEED = 42

X, y = make_moons(n_samples=NUM_SAMPLES,
#menambahkan noise ke dataset untuk membuatnya lebih realistis
                  noise=0.07,
#mengontrol random seed sehingga hasil yang dihasilkan dapat di
reproduksi
                  random_state=RANDOM_SEED)

#hasil adalah 2 array
X[:10], y[:10]
```

Output

```
(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]))
```

Input

```
# Turn data into a DataFrame
import pandas as pd
#membuat dataframe baru
data_df = pd.DataFrame({"X0": X[:, 0],
                        "X1": X[:, 1],
                        "y": y})
#menampilkan lima baris pertama dari dataframe
data_df.head()
```

Output

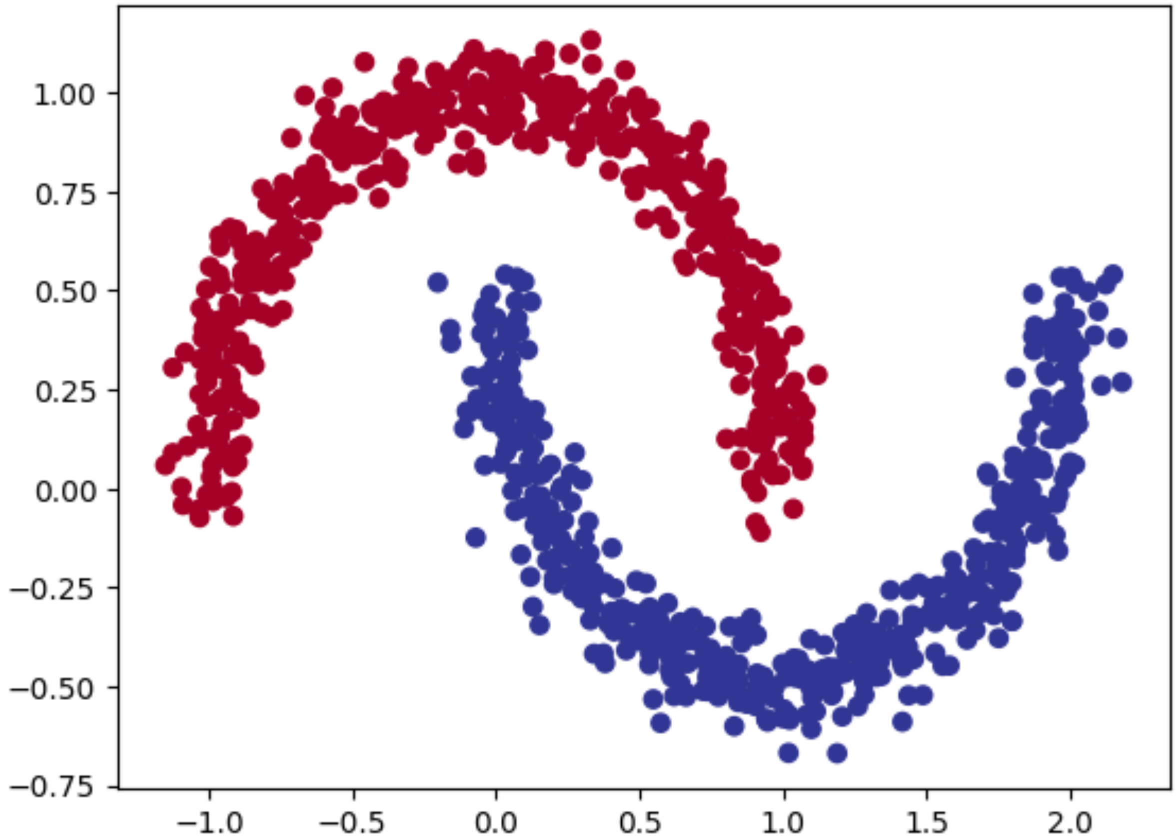
	X0	X1	y
0	-0.033411	0.421391	1
1	0.998827	-0.442890	1
2	0.889592	-0.327843	1
3	0.341958	-0.417690	1

	X0	X1	y
4	-0.838531	0.532375	0

Input

```
# Visualize the data on a plot
import matplotlib.pyplot as plt
#membuat scatter plot
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu);
```

Output



Input

```
# Turn data into tensors
#mengubah array menjadi tensor PyTorch dengan tipe data float
X = torch.tensor(X, dtype=torch.float)
y = torch.tensor(y, dtype=torch.float)

# Split the data into train and test sets
from sklearn.model_selection import train_test_split
#data yang sudah di konversi ke dalam tensor dibagi menjadi data
latihan
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
```

```
random_state=RANDOM_SEED)

#menghitung dan mencetak panjang sampel
len(X_train), len(X_test), len(y_train), len(y_test)
```

Output

(800, 200, 800, 200)

2. Build a model by subclassing nn.Module that incorporates non-linear activation functions and is capable of fitting the data you created in 1.

Input

```
import torch
from torch import nn

class MoonModelV0(nn.Module):
    def __init__(self, in_features, out_features, hidden_units):
        super().__init__()

        self.layer1 = nn.Linear(in_features=in_features,
                                 out_features=hidden_units)
        self.layer2 = nn.Linear(in_features=hidden_units,
                                 out_features=hidden_units)
        self.layer3 = nn.Linear(in_features=hidden_units,
                                out_features=out_features)
        self.relu = nn.ReLU()

    def forward(self, x):
        return
self.layer3(self.relu(self.layer2(self.relu(self.layer1(x)))))

#menentukan perangkat (GPU atau CPU) yang akan digunakan model
model_0 = MoonModelV0(in_features=2,
                      out_features=1,
                      hidden_units=10).to(device)
#membuat model dalam pernagkat yang akan dipilih secara default oleh
PyTorch
model_0
```

Output

```
MoonModelV0(
  (layer1): Linear(in_features=2, out_features=10, bias=True)
  (layer2): Linear(in_features=10, out_features=10, bias=True)
  (layer3): Linear(in_features=10, out_features=1, bias=True)
  (relu): ReLU()
)
```

Input

```
#mengembalikan dictionary yang berisi informasi lengkap mengenai nilai
dari setiap parameter
model_0.state_dict()
```

Output

```
OrderedDict([('layer1.weight',
          tensor([[-0.1347,  0.5775],
                [ 0.2925,  0.6565],
                [ 0.4486, -0.3270],
                [-0.5363,  0.5108],
                [ 0.4884,  0.1700],
                [ 0.1675,  0.0717],
                [-0.5877, -0.4342],
                [-0.2554, -0.2677],
                [-0.4260,  0.4323],
                [ 0.0798, -0.1119]], device='cuda:0')),
         ('layer1.bias',
          tensor([-0.4645, -0.0870, -0.6391, -0.6704,  0.1283, -0.6653, -0.5581, -0.1774,
                -0.4731, -0.1441], device='cuda:0')),
         ('layer2.weight',
          tensor([[-0.0503,  0.0808, -0.2741,  0.1574, -0.0555,  0.1186,  0.0897, -0.2431,
                  0.0529,  0.0201],
                [ 0.1784, -0.2360,  0.1742, -0.1766,  0.0180,  0.2770,  0.0170,  0.0299,
                 -0.1775, -0.3096],
                [-0.1521, -0.2032,  0.0648, -0.1277, -0.2515, -0.0269, -0.1312,  0.1273,
                  0.1107,  0.3136],
                [-0.1946, -0.1397, -0.1615,  0.2797,  0.0061,  0.3020, -0.1746,  0.2781,
                 -0.3088, -0.1882],
                [ 0.0755, -0.2294, -0.2819,  0.1481,  0.0649, -0.0813,  0.0808,  0.1799,
                  0.0592, -0.2220],
                [-0.2044,  0.1902,  0.1227,  0.2999, -0.2625, -0.2235, -0.1266, -0.0901,
                 -0.1551,  0.2209],
                [-0.1314,  0.0521, -0.0004, -0.0906,  0.2046,  0.2129, -0.3017, -0.2074,
                 -0.2223,  0.2468],
                [ 0.1992, -0.1483, -0.2583,  0.1499,  0.2329,  0.0769, -0.1091, -0.1149,
                 -0.1156,  0.2711],
                [-0.0437,  0.1830, -0.0476,  0.1019,  0.0301, -0.1945, -0.1671,  0.1885,
                 -0.1632, -0.2724],
                [-0.2491,  0.0923,  0.2736,  0.0329,  0.0655,  0.0320,  0.2737, -0.0257,
                 -0.1923, -0.1205]], device='cuda:0')),
         ('layer2.bias',
          tensor([-0.2431, -0.1266,  0.1779,  0.0580, -0.0605,  0.0248,  0.1016,  0.3067,
                -0.2753,  0.1095], device='cuda:0')),
         ('layer3.weight',
          tensor([[-0.1343, -0.2595, -0.0045,  0.0350, -0.2345, -0.0764, -0.1587, -0.2079,
                 -0.1677, -0.2727]], device='cuda:0')),
         ('layer3.bias', tensor([-0.1380], device='cuda:0'))])
```

3. Setup a binary classification compatible loss function and optimizer to use when training the model built in 2.

Input

```
loss_fn = nn.BCEWithLogitsLoss() # sigmoid layer built-in
# loss_fn = nn.BCELoss() # requires sigmoid layer
optimizer = torch.optim.SGD(params=model_0.parameters(), # parameters of model to optimize
                            lr=0.1) # learning rate
```

4. Create a training and testing loop to fit the model you created in 2 to the data you created in 1.

Input

```
# What's coming out of our model?

#mencetak keluaran model
print("Logits:")
#menghilangkan dimensi yang tidak perlu
print(model_0(X_train.to(device)[:10]).squeeze())

#memberikan probabilitas setiap sampel termasuk ke dalam kelas positif
print("Pred probs:")
#menghilangkan dimensi yang tidak perlu
print(torch.sigmoid(model_0(X_train.to(device)[:10]).squeeze()))

#memberikan label prediksi biner (0 atau 1) untuk setiap sampel
print("Pred labels:")
#menghilangkan dimensi yang tidak perlu
print(torch.round(torch.sigmoid(model_0(X_train.to(device)[:10]).squeez
e())))
```

Output

```
Logits:
tensor([-0.3673, -0.2363, -0.2639, -0.2708, -0.2732, -0.2637, -0.2807,
-0.2662,
        -0.3022, -0.2463], device='cuda:0', grad_fn=<SqueezeBackward0>)
Pred probs:
tensor([0.4092, 0.4412, 0.4344, 0.4327, 0.4321, 0.4345, 0.4303, 0.4338,
0.4250,
        0.4387], device='cuda:0', grad_fn=<SigmoidBackward0>)
Pred labels:
tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], device='cuda:0',
       grad_fn=<RoundBackward0>)
```

Input

```
# Let's calculate the accuracy
!pip -q install torchmetrics # colab doesn't come with torchmetrics
#menghitung akurasi dan dipindahkan ke perangkat sebelumnya
from torchmetrics import Accuracy
acc_fn = Accuracy(task="multiclass", num_classes=2).to(device) # send
accuracy function to device
acc_fn
```

Output

806.1/806.1 kB 4.4 MB/s eta 0:00:00
MulticlassAccuracy()

Input

```
torch.manual_seed(RANDOM_SEED)

epochs=1000
```

```
# Send data to the device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

# Loop through the data
for epoch in range(epochs):
  ### Training
  model_0.train()

  # 1. Forward pass
  y_logits = model_0(X_train).squeeze()
  # print(y_logits[:5]) # model raw outputs are "logits"
  y_pred_probs = torch.sigmoid(y_logits)
  y_pred = torch.round(y_pred_probs)

  # 2. Calculaute the loss
  loss = loss_fn(y_logits, y_train) # loss = compare model raw outputs
to desired model outputs
  acc = acc_fn(y_pred, y_train.int()) # the accuracy function needs to
compare pred labels (not logits) with actual labels

  # 3. Zero the gradients
  optimizer.zero_grad()

  # 4. Loss backward (perform backpropagation) -
https://brilliant.org/wiki/backpropagation/#:~:text=Backpropagation%2C%
20short%20for%20%22backward%20propagation,to%20the%20neural%20network's
%20weights.
  loss.backward()

  # 5. Step the optimizer (gradient descent) -
https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-
cf04e8115f21#:~:text=Gradient%20descent%20(GD)%20is%20an,e.g.%20in%20a%
20linear%20regression)
  optimizer.step()

  ### Testing
  model_0.eval()
  with torch.inference_mode():
    # 1. Forward pass
    test_logits = model_0(X_test).squeeze()
    test_pred = torch.round(torch.sigmoid(test_logits))
    # 2. Caculate the loss/acc
    test_loss = loss_fn(test_logits, y_test)
    test_acc = acc_fn(test_pred, y_test.int())

  # Print out what's happening
```

```
  if epoch % 100 == 0:
    print(f"Epoch: {epoch} | Loss: {loss:.2f} Acc: {acc:.2f} | Test
loss: {test_loss:.2f} Test acc: {test_acc:.2f}")
```

Output

```
Epoch: 0 | Loss: 0.71 Acc: 0.50 | Test loss: 0.71 Test acc: 0.50
Epoch: 100 | Loss: 0.67 Acc: 0.86 | Test loss: 0.67 Test acc: 0.87
Epoch: 200 | Loss: 0.38 Acc: 0.87 | Test loss: 0.39 Test acc: 0.86
Epoch: 300 | Loss: 0.25 Acc: 0.88 | Test loss: 0.25 Test acc: 0.89
Epoch: 400 | Loss: 0.24 Acc: 0.89 | Test loss: 0.24 Test acc: 0.90
Epoch: 500 | Loss: 0.23 Acc: 0.90 | Test loss: 0.23 Test acc: 0.89
Epoch: 600 | Loss: 0.22 Acc: 0.90 | Test loss: 0.22 Test acc: 0.89
Epoch: 700 | Loss: 0.22 Acc: 0.90 | Test loss: 0.22 Test acc: 0.89
Epoch: 800 | Loss: 0.21 Acc: 0.90 | Test loss: 0.21 Test acc: 0.89
Epoch: 900 | Loss: 0.21 Acc: 0.91 | Test loss: 0.21 Test acc: 0.89
```

5. Make predictions with your trained model and plot them using the plot_decision_boundary() function created in this notebook.

Input

```
# Plot the model predictions

import numpy as np

# TK - this could go in the helper_functions.py and be explained there
def plot_decision_boundary(model, X, y):

    # mengubah model dan data dari pernagkat CUDA ke CPU
    model.to("cpu")
    X, y = X.to("cpu"), y.to("cpu")

    # Source - https://madewithml.com/courses/foundations/neural-
networks/
    # membuat grid untuk plotting decision boundary dengan menggunakan
minimum dan maksimum dari data dalam setiap dimensi
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 101),
                         np.linspace(y_min, y_max, 101))

    # menyiapkan data yang akan digunakan untuk melakukan prediksi
decision boundary
    X_to_pred_on = torch.from_numpy(np.column_stack((xx.ravel(),
yy.ravel()))).float()

    # melakukan prediksi pada data yang telah disiapkan
    model.eval()
    with torch.inference_mode():
        y_logits = model(X_to_pred_on)

    # menyesuaikan keluaran model ke bentuk label prediksi
    if len(torch.unique(y)) > 2:
```

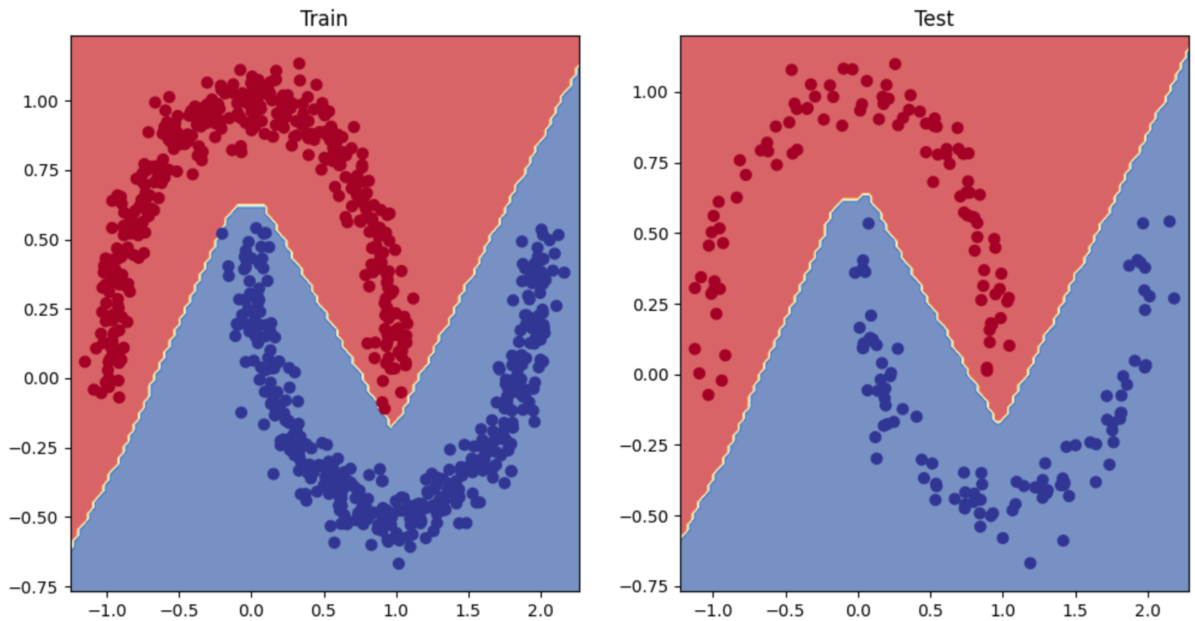
```
        y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1) # mutli-
class
    else:
        y_pred = torch.round(torch.sigmoid(y_logits)) # binary

    # menggambar decision boundary
    y_pred = y_pred.reshape(xx.shape).detach().numpy()
    # menampilkan data asli
    plt.contourf(xx, yy, y_pred, cmap=plt.cm.RdYlBu, alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
```

Input

```
# membuat gambar matplotlib baru
plt.figure(figsize=(12, 6))
# membuat subplot pertama
plt.subplot(1, 2, 1)
# menambahkan judul "Train" pada subplot pertama
plt.title("Train")
# memanggil fungsi
plot_decision_boundary(model_0, X_train, y_train)
# membuat subplpot kedua
plt.subplot(1, 2, 2)
# menambahkan judul "Test" pada subplot kedua
plt.title("Test")
# membuat decision boundary dari data uji dan menampikannya pada
subplot kedua
plot_decision_boundary(model_0, X_test, y_test)
```

Output



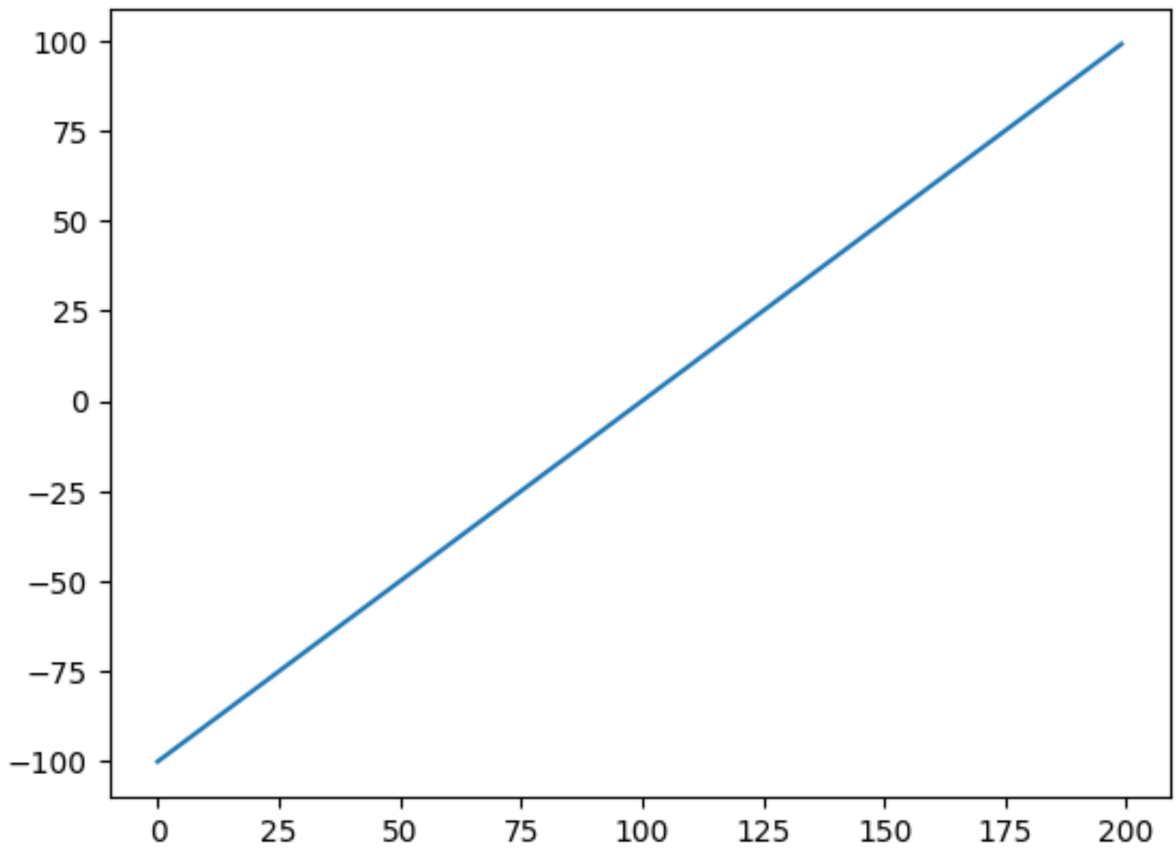
6. Replicate the Tanh (hyperbolic tangent) activation function in pure PyTorch.

Input

```
# menunjukkan nilai dalam tensor terhadap indeksnya
tensor_A = torch.arange(-100, 100, 1)
# menggambarkan nilai terhadap indeksnya dari 0 hingga panangg tensor
plt.plot(tensor_A)
```

Output

```
[<matplotlib.lines.Line2D at 0x7b0f10b42a10>]
```

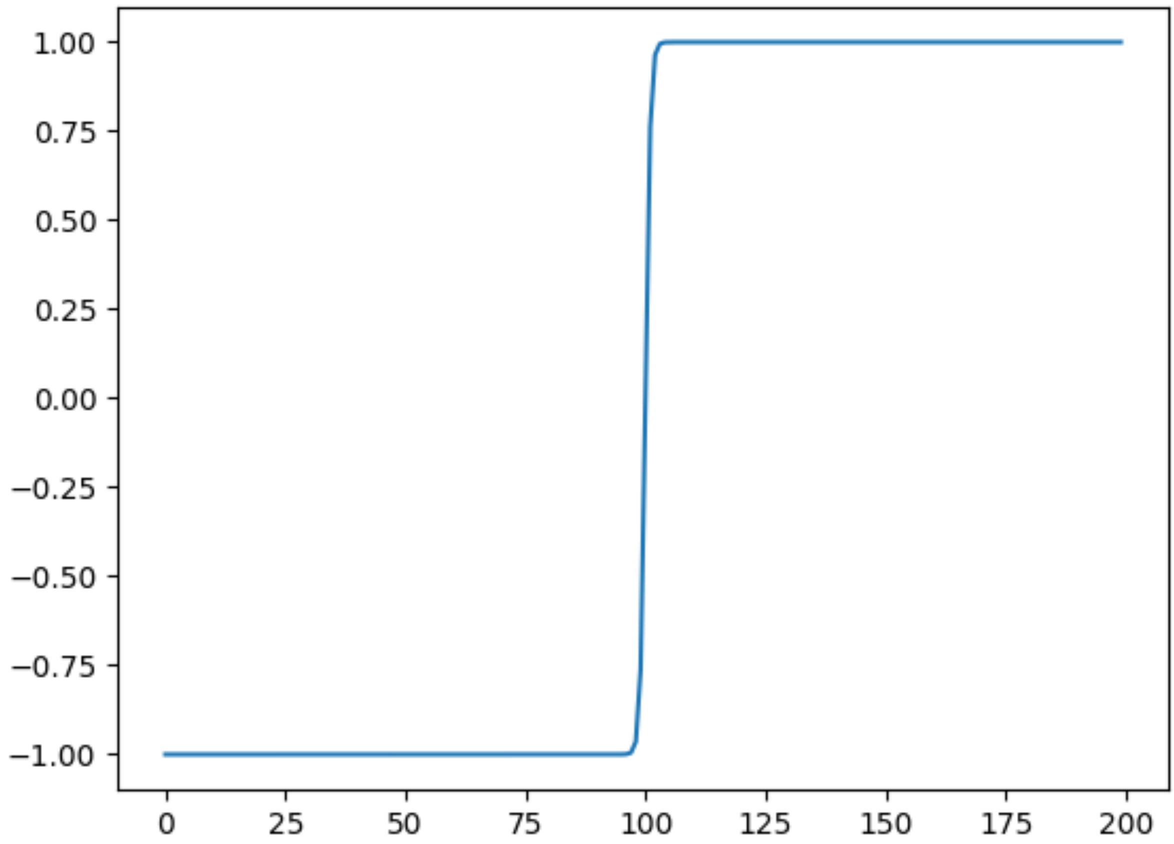


Input

```
# grafik yang dihasilkan akan menunjukkan nilai yang dihasilkan oleh
fungsi tangen hiperbolik terhadap nilai tensor
plt.plot(torch.tanh(tensor_A))
```

Output

```
[<matplotlib.lines.Line2D at 0x7b0f10bd2fb0>]
```



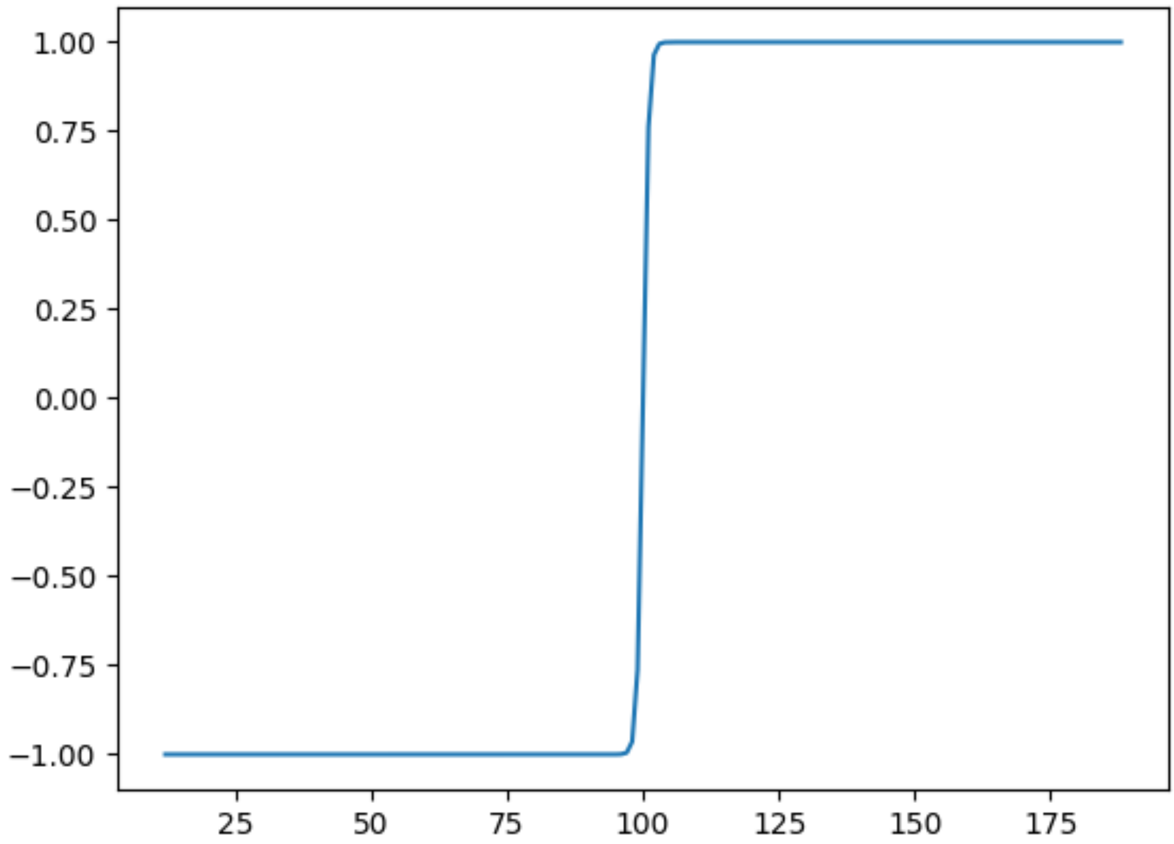
Input

```
def tanh(x):
  # Source - https://ml-
cheatsheet.readthedocs.io/en/latest/activation_functions.html#tanh
  return (torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-
x))

plt.plot(tanh(tensor_A))
```

Output

```
[<matplotlib.lines.Line2D at 0x7b0f10e3c880>]
```

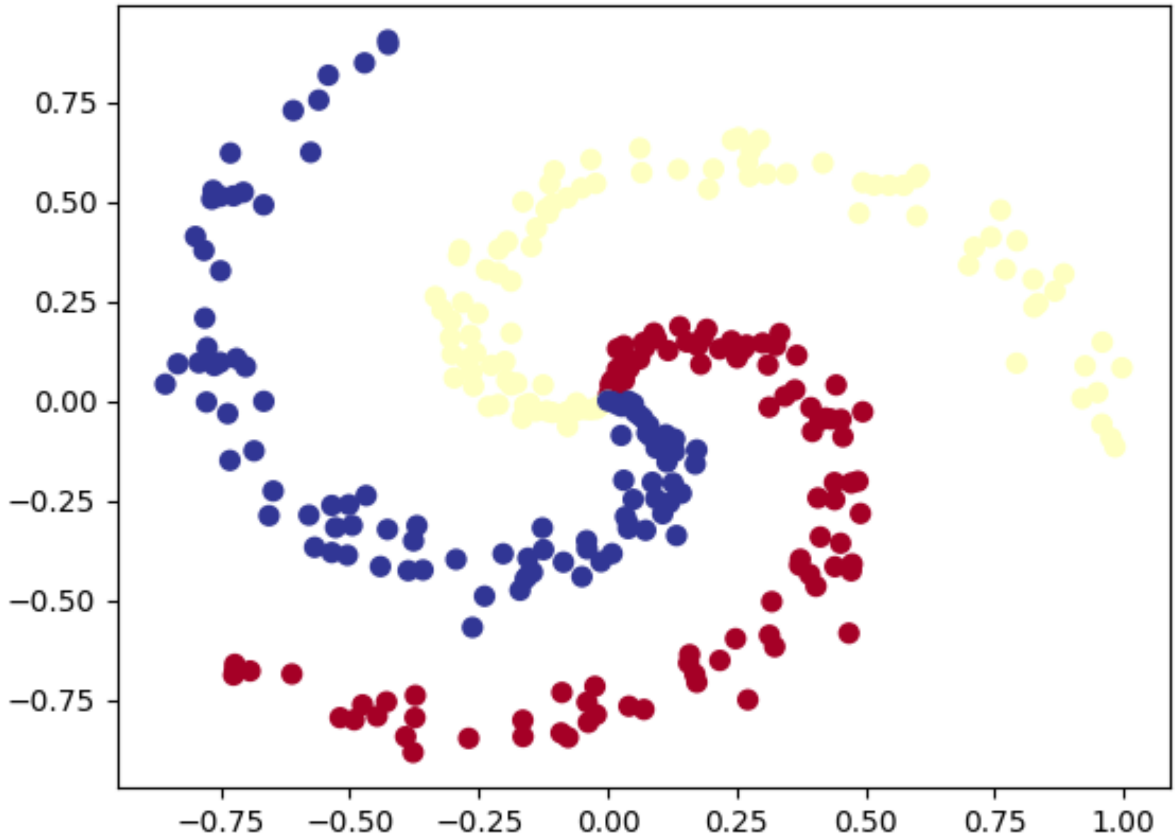


7. Create a multi-class dataset using the spirals data creation function from CS231n (see below for the code).

Input

```
# Code for creating a spiral dataset from CS231n
import numpy as np
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
N = 100 # jumlah titik data per kelas
D = 2 # dimensi dari setiap titik data
K = 3 # jumlah kelas
X = np.zeros((N*K,D)) # matriks data
y = np.zeros(N*K, dtype='uint8') # label kelas untuk setiap titik data
for j in range(K):
  ix = range(N*j,N*(j+1))
# menghasilkan bilangan teratur antara 0 dan 1, mempresentasikan radius
dari spiral
  r = np.linspace(0.0,1,N)
# menghasilkan nilai sudut yang berkembang dari 0 hingga 4 per kelas
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2
# mengubah koordinat polar menjadi koordinat kartesian
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
# memberikan label kelas kepada setiap titik data
  y[ix] = j
# lets visualize the data
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)
plt.show()
```

Output



Input

```
# Turn data into tensors
X = torch.from_numpy(X).type(torch.float) # features as float32
y = torch.from_numpy(y).type(torch.LongTensor) # labels need to be of
type long

# Create train and test splits
from sklearn.model_selection import train_test_split
# membagi data ke dalam subset data latihan dan subset data uji
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=RANDOM_SEED)
# menghitung dan mencetak panjang dari umlah sampel
len(X_train), len(X_test), len(y_train), len(y_test)
```

Output

(240, 60, 240, 60)

Input

```
# Let's calculate the accuracy for when we fit our model
!pip -q install torchmetrics # colab doesn't come with torchmetrics
from torchmetrics import Accuracy
# menetapkan perhitungan akurasi untuk masalah klasifikasi multikelas
serta memindahkan objek ke perangkat yang telah ditentukan
acc_fn = Accuracy(task="multiclass", num_classes=3).to(device)
# mengukur akurasi dari prediksi pada data uji
acc_fn
```

Output
MulticlassAccuracy()

Input
```
# Prepare device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"

class SpiralModel(nn.Module):
  def __init__(self):
    super().__init__()
# self.linear adalah tiga linear berturut-turut dalam model,
# in_features adalah jumlah fitur masukan ke layer
# out_features adalah menentukan jumlah unit / dimensi output dari
setiap layer
    self.linear1 = nn.Linear(in_features=2, out_features=10)
    self.linear2 = nn.Linear(in_features=10, out_features=10)
    self.linear3 = nn.Linear(in_features=10, out_features=3)
# aktivasi ReLU yang diterapkan di antara setiap layer linear
    self.relu = nn.ReLU()

# aliran / urutan dari setiap layer dalam model saat menerima input 'x'
  def forward(self, x):
    return
self.linear3(self.relu(self.linear2(self.relu(self.linear1(x)))))

# memindahkan model ke perangkat yang tersedia
model_1 = SpiralModel().to(device)
model_1
```
Output
```
SpiralModel(
  (linear1): Linear(in_features=2, out_features=10, bias=True)
  (linear2): Linear(in_features=10, out_features=10, bias=True)
  (linear3): Linear(in_features=10, out_features=3, bias=True)
  (relu): ReLU()
)
```

Input
```
# Setup data to be device agnostic
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)
print(X_train.dtype, X_test.dtype, y_train.dtype, y_test.dtype)

# Print out untrained model outputs
print("Logits:")
print(model_1(X_train)[:10])

print("Pred probs:")
print(torch.softmax(model_1(X_train)[:10], dim=1))

print("Pred labels:")
```

```
print(torch.softmax(model_1(X_train)[:10], dim=1).argmax(dim=1))
```
Output

```
torch.float32 torch.float32 torch.int64 torch.int64
Logits:
tensor([[-0.2160, -0.0600,  0.2256],
        [-0.2020, -0.0530,  0.2257],
        [-0.2223, -0.0604,  0.2384],
        [-0.2174, -0.0555,  0.2826],
        [-0.2201, -0.0502,  0.2792],
        [-0.2195, -0.0565,  0.2457],
        [-0.2212, -0.0581,  0.2440],
        [-0.2251, -0.0631,  0.2354],
        [-0.2116, -0.0548,  0.2336],
        [-0.2170, -0.0552,  0.2842]], device='cuda:0',
       grad_fn=<SliceBackward0>)
Pred probs:
tensor([[0.2685, 0.3139, 0.4176],
        [0.2707, 0.3142, 0.4151],
        [0.2659, 0.3126, 0.4215],
        [0.2615, 0.3074, 0.4311],
        [0.2609, 0.3092, 0.4299],
        [0.2653, 0.3123, 0.4224],
        [0.2653, 0.3123, 0.4224],
        [0.2659, 0.3127, 0.4214],
        [0.2681, 0.3136, 0.4184],
        [0.2614, 0.3072, 0.4314]], device='cuda:0', grad_fn=<SoftmaxBackward0>)
Pred labels:
tensor([2, 2, 2, 2, 2, 2, 2, 2, 2, 2], device='cuda:0')
```

Input

```
# meminimalkan perbedaan antara distribusi probabilitas prediksi model
dan label dari data
loss_fn = nn.CrossEntropyLoss()
# menentukan paramaeter yang akan di optimalkan oleh optimazer
optimizer = torch.optim.Adam(model_1.parameters(),
                             lr=0.02)
```

Input

```
# Build a training loop for the model
epochs = 1000

# Loop over data
for epoch in range(epochs):
  ## Training
  model_1.train()
  # 1. forward pass
  # melakukan perhitungan output model untuk data latihan dan data uji
  y_logits = model_1(X_train)
  # menghasilkan prediksi kelas dengan memilih kelas dengan
probabilitas tertinggi dari output model
  y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1)

  # 2. calculate the loss
```

```
  # menghitung loss untuk data latihan dan data uji berdasarkan output
model dan label yang sebenarnya
  loss = loss_fn(y_logits, y_train)
  # menghitung akurasi untuk data latihan dan data uji berdasarkan
prediksi model dan label yang sebenarnya
  acc = acc_fn(y_pred, y_train)

  # 3. optimizer zero grad
  # mengosongkan gradien pada parameter model sebelum melakukan
backpropaation
  optimizer.zero_grad()

  # 4. loss backwards
  # melakukan backpropagation untuk menghitung gradien loss terhadap
parameter model
  loss.backward()

  # 5. optimizer step step step
  # melakukan lankah optimasi untuk mengupdate parameter model
berdasarkan gradien yang dihitung sebelumnya
  optimizer.step()

  ## Testing
  model_1.eval()
  with torch.inference_mode():
    # 1. Forward pass
    test_logits = model_1(X_test)
    test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
    # 2. Caculate loss and acc
    test_loss = loss_fn(test_logits, y_test)
    test_acc = acc_fn(test_pred, y_test)

  # Print out what's happening
  if epoch % 100 == 0:
    print(f"Epoch: {epoch} | Loss: {loss:.2f} Acc: {acc:.2f} | Test
loss: {test_loss:.2f} Test acc: {test_acc:.2f}")
```

Output

```
Epoch: 0 | Loss: 1.12 Acc: 0.32 | Test loss: 1.10 Test acc: 0.37
Epoch: 100 | Loss: 0.45 Acc: 0.78 | Test loss: 0.53 Test acc: 0.68
Epoch: 200 | Loss: 0.12 Acc: 0.96 | Test loss: 0.09 Test acc: 0.98
Epoch: 300 | Loss: 0.07 Acc: 0.98 | Test loss: 0.02 Test acc: 1.00
Epoch: 400 | Loss: 0.05 Acc: 0.98 | Test loss: 0.01 Test acc: 1.00
Epoch: 500 | Loss: 0.04 Acc: 0.99 | Test loss: 0.01 Test acc: 1.00
Epoch: 600 | Loss: 0.03 Acc: 0.99 | Test loss: 0.01 Test acc: 1.00
Epoch: 700 | Loss: 0.03 Acc: 0.99 | Test loss: 0.00 Test acc: 1.00
Epoch: 800 | Loss: 0.02 Acc: 0.99 | Test loss: 0.00 Test acc: 1.00
Epoch: 900 | Loss: 0.02 Acc: 0.99 | Test loss: 0.00 Test acc: 1.00
```

Input

```
# Plot decision boundaries for training and test sets
```

```
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)
```

Output