```
#Nama: Dilara Kynta Putri Raflita
#NIM: 1103204059
#Kelas: TK44G4 Machine Learning
```

02. PyTorch Classification Exercise

```
# Check for GPU
!nvidia-smi
```

```
Wed Jan  3 14:46:51 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   50C    P8              12W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

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

pada kode tersebut digunakan untuk memeriksa apakah sistem memiliki GPU NVIDIA yang terdeteksi dan untuk melihat informasi terkait kinerja dan penggunaan GPU secara real-time.

```
# Import torch
import torch

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

```
'cuda'
```

pada kode tersebut digunakan untuk memeriksa apakah CUDA tersedia atau tidak dengan menggunakan

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

- For consistency, the dataset should have 1000 samples and a `random_state=42`.
- Turn the data into PyTorch tensors.
- Split the data into training and test sets using `train_test_split` with 80% training and 20% testing.

```
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]))
```

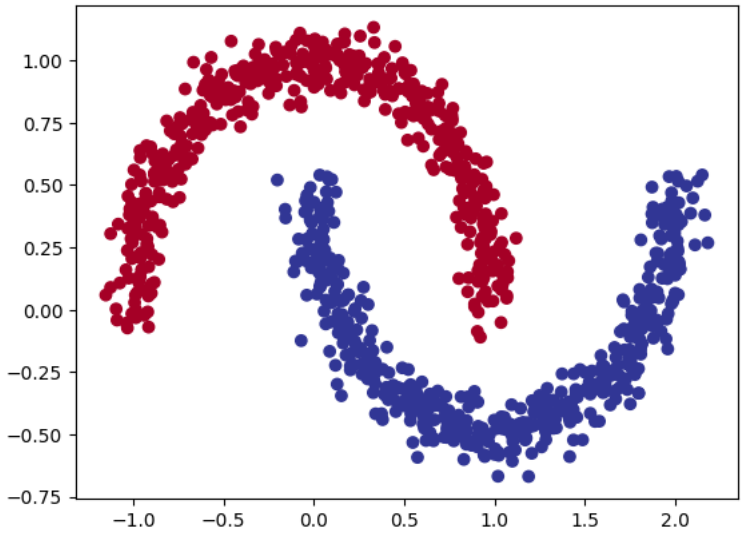
kode tersebut menggunakan fungsi 'make_moons' dari library scikit-learn untuk menghasikan dataset dummy. pada perintah X[:10], y:[10} adalah vektor target yang berisi label kelas untuk setiap sampel. output yang diberikan adalah sepuluh baris pertama yang dicetak untuk memberikan gambaran awal tentang isi dataset.

```
# Turn data into a DataFrame
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

dataset yang telah dihasilkan sebelumnya diubah menjadi objek dataframe menggunakan library pandas. pada perintah data_df = pd.DataFrame({"X0": X[:, 0], "X1": X[:, 1], "y": y}) mengambil kolom pertama dan kedua dari matriks fitur ;X;

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



kode tersebut digunakan untuk membuat scatter plot yang memvisualisasikan dataset yang telah dibuat sebelumnya. hasil visualisasi menunjukkan sebaran data pada ruang fitur dua dimensi, dimana sumbu x dan y mewakili dua fitur dari dataset. pemisahan antara dua kelas mencerminkan pola bulan sabit yang dihasilkan oleh fungsi 'make_moons'.

```
# Turn data into tensors
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
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=RANDOM_SEED)

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

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

Output ini memberikan informasi tentang bagaimana data terbagi untuk proses pelatihan dan evaluasi model. jumlah sampel dalam set pelatihan akan menjadi sekitar 80% dari total sampel, sedangkan jumlah sampel dalam set pengujian akan menjadi sekitar 20%

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.

- Feel free to use any combination of PyTorch layers (linear and non-linear) you want.

```
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)))))

model_0 = MoonModelV0(in_features=2,
                      out_features=1,
                      hidden_units=10).to(device)
model_0
```

```
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()
)
```

kode tersebut mendefinisikan bagaimana model digunakan untuk mempelajari pola pada dataset dua dimensi yang sudah dihasilkan sebelumnya

```
model_0.state_dict()
```

```
OrderedDict([('layer1.weight',
              tensor([[-0.6671,  0.5939],
                      [-0.2115, -0.2905],
                      [-0.4585,  0.2137],
                      [-0.2600,  0.3245],
                      [ 0.2904,  0.3980],
                      [-0.4750, -0.3012],
                      [ 0.0073,  0.5940],
                      [ 0.6575,  0.2451],
                      [ 0.5579, -0.2432],
                      [-0.0212,  0.2427]], device='cuda:0')),
             ('layer1.bias',
              tensor([ 0.5002, -0.3500,  0.6639, -0.3904, -0.6309, -0.0412, -0.5563,  0.5985,
                      -0.2839, -0.6206], device='cuda:0')),
             ('layer2.weight',
              tensor([[-0.2289, -0.2952, -0.1622, -0.1027, -0.1782,  0.0678,  0.1052,  0.1773,
                       -0.3115, -0.0418],
                      [ 0.2017, -0.3055,  0.0949,  0.0344, -0.1977, -0.0831,  0.2007, -0.0677,
                       -0.3087, -0.1807],
                      [-0.0308,  0.1236, -0.2116, -0.1797,  0.2666, -0.1262, -0.1391, -0.0637,
                       -0.0228,  0.2051],
                      [ 0.2360, -0.0033,  0.1229, -0.0559,  0.2874,  0.0942, -0.1401, -0.1138,
                       -0.2266,  0.0062],
                      [-0.2427, -0.0819, -0.0653,  0.1223,  0.2411, -0.0572,  0.0762, -0.2197,
                       -0.2422,  0.2851],
                      [ 0.2972,  0.1754,  0.1882, -0.2540, -0.0838, -0.0512,  0.0521,  0.1352,
                       -0.2880, -0.1327],
                      [-0.2248,  0.1007, -0.1713, -0.2165,  0.2472, -0.1098,  0.1207,  0.1648,
                       -0.1866,  0.2828],
                      [ 0.2425, -0.2169, -0.0149,  0.1111,  0.3115, -0.2230, -0.2196,  0.1738,
                       -0.2175, -0.0142],
                      [-0.1064,  0.0711,  0.0459, -0.1346,  0.0120,  0.2344,  0.0624,  0.2044,
                       -0.2721, -0.3151],
                      [-0.1405,  0.2857,  0.1756,  0.1924, -0.3102, -0.0692, -0.1834, -0.0476,
                        0.1452, -0.2881]], device='cuda:0')),
             ('layer2.bias',
              tensor([-0.1359, -0.1311,  0.0586, -0.1558, -0.1019, -0.0463,  0.2979, -0.2700,
                       0.2482, -0.2758], device='cuda:0')),
             ('layer3.weight',
              tensor([[-0.2873,  0.3059,  0.2613,  0.1587, -0.2122, -0.0430, -0.0380,  0.0344,
                        0.1492,  0.1216]], device='cuda:0')),
             ('layer3.bias', tensor([0.0692], device='cuda:0'))])
```

pada perintah model_0.state_dict() digunakan untuk mengakses dan mencetak nilai-nilai state parameter dari model. hasil output tersebut yaitu suatu dictionary yang memetakan nama setiap parameter pada nilai tensor yang sesuai. state dictionary ini digunakan untuk menyimpan semua informasi yang dibutuhkan untuk mengembalikan model ke keadaan yang sama seperti pada saat penyimpanan

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

```
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
```

kode tersebut memberi perintah untuk menginisialisasi fungsi kerugian sebagai 'BCEWithLogitsLoss' yang digunakan untuk permasalahan biner klasifikasi yang mana output dari model tidak diaktivasi oleh fungsi sigmoid

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

- To measure model accuray, you can create your own accuracy function or use the accuracy function in TorchMetrics.
- Train the model for long enough for it to reach over 96% accuracy.
- The training loop should output progress every 10 epochs of the model's training and test set loss and accuracy.

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

# logits (raw outputs of model)
print("Logits:")
print(model_0(X_train.to(device)[:10]).squeeze())

# Prediction probabilities
print("Pred probs:")
print(torch.sigmoid(model_0(X_train.to(device)[:10]).squeeze()))

# Prediction probabilities
print("Pred labels:")
print(torch.round(torch.sigmoid(model_0(X_train.to(device)[:10]).squeeze())))
```

```
Logits:
tensor([0.1164, 0.2091, 0.1704, 0.1624, 0.1116, 0.1672, 0.1125, 0.1043, 0.1105,
        0.1934], device='cuda:0', grad_fn=<SqueezeBackward0>)
Pred probs:
tensor([0.5291, 0.5521, 0.5425, 0.5405, 0.5279, 0.5417, 0.5281, 0.5261, 0.5276,
        0.5482], device='cuda:0', grad_fn=<SigmoidBackward0>)
Pred labels:
tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], device='cuda:0',
       grad_fn=<RoundBackward0>)
```

output tersebut memberikan gambaran tentang bagaimana model memberikan prediksi pada sepuluh sampel pertama dari set pelatihan.

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

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 806.1/806.1 kB 13.8 MB/s eta 0:00:00
MulticlassAccuracy()
```

```
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()
```

```
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%
loss.backward()

# 5. Step the optimizer (gradient descent) - https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21#:~:text
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}")
```

```
 Epoch: 0 | Loss: 0.71 Acc: 0.50 | Test loss: 0.70 Test acc: 0.50
 Epoch: 100 | Loss: 0.36 Acc: 0.82 | Test loss: 0.38 Test acc: 0.80
 Epoch: 200 | Loss: 0.26 Acc: 0.88 | Test loss: 0.26 Test acc: 0.86
 Epoch: 300 | Loss: 0.24 Acc: 0.88 | Test loss: 0.23 Test acc: 0.90
 Epoch: 400 | Loss: 0.23 Acc: 0.89 | Test loss: 0.22 Test acc: 0.90
 Epoch: 500 | Loss: 0.21 Acc: 0.89 | Test loss: 0.21 Test acc: 0.92
 Epoch: 600 | Loss: 0.19 Acc: 0.91 | Test loss: 0.18 Test acc: 0.94
 Epoch: 700 | Loss: 0.16 Acc: 0.93 | Test loss: 0.15 Test acc: 0.94
 Epoch: 800 | Loss: 0.11 Acc: 0.95 | Test loss: 0.11 Test acc: 0.96
 Epoch: 900 | Loss: 0.08 Acc: 0.98 | Test loss: 0.07 Test acc: 0.99
```

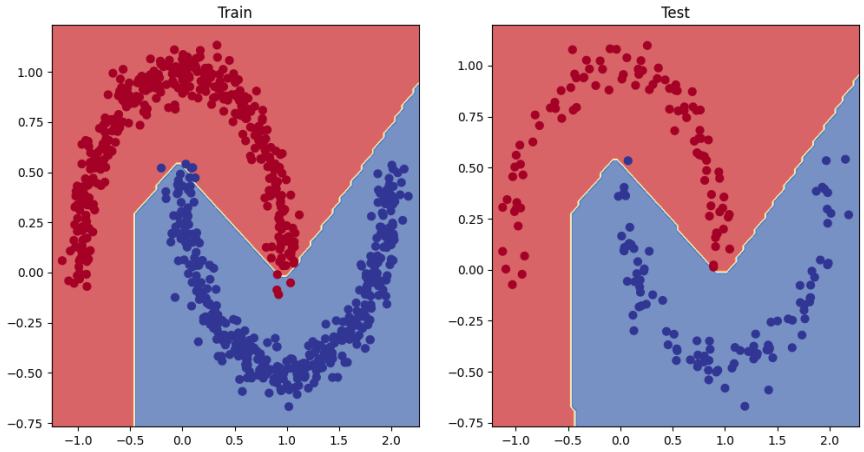
kode tersebut memberi perintah untuk menghitung akurasi dari prediksi yang dihasilkan oleh model. output yang dihasilkan memberikan informasi tentang parameter dan konfigurasi fungsi akurasi yang akan digunakan

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

```
# 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):

    # Put everything to CPU (works better with NumPy + Matplotlib)
    model.to("cpu")
    X, y = X.to("cpu"), y.to("cpu")

    # Source - https://madewithml.com/courses/foundations/neural-networks/
    # (with modifications)
    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))

    # Make features
    X_to_pred_on = torch.from_numpy(np.column_stack((xx.ravel(), yy.ravel()))).float()

    # Make predictions
    model.eval()
    with torch.inference_mode():
        y_logits = model(X_to_pred_on)

    # Test for multi-class or binary and adjust logits to prediction labels
    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

    # Reshape preds and plot
    y_pred = y_pred.reshape(xx.shape).detach().numpy()
    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())
```

kode tersebut berfungsi untuk memvisualisasikan bagaimana nanti model mengklasifikasikan data dalam ruang fitur dua dimensi dengan menggambar batas keputusan

```
# 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_0, X_train, y_train)
plt.subplot(1, 2, 2)
plt.title("Test")
plot_decision_boundary(model_0, X_test, y_test)
```



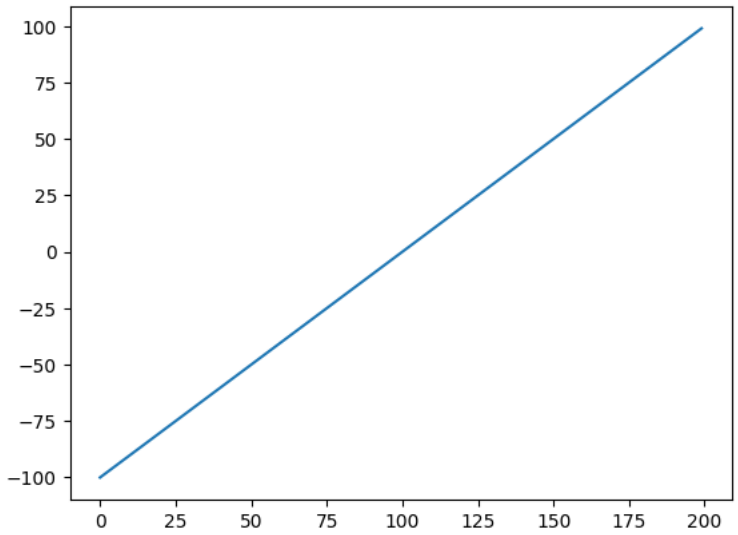
kode tersebut digunakan untuk membuat dua buah subplot yang menunjukkan batas keputusan dari model pada set pelatihan dan set pengujian. hasil output tersebut menunjukkan adanya dua subplot sejajar yang menunjukkan batas keputusan dari model. sementara titik titik pada plot menunjukkan distribusi data asli dengan warna sesuai dengan label kelas.

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

- Feel free to reference the ML cheatsheet website for the formula.

```
tensor_A = torch.arange(-100, 100, 1)
plt.plot(tensor_A)
```

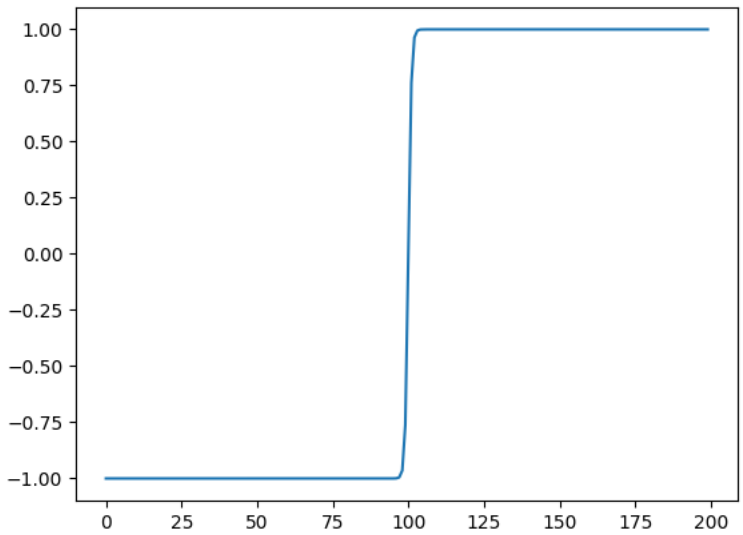
```
[<matplotlib.lines.Line2D at 0x7ae7efb91540>]
```



kode tersebut digunakan untuk membuat plot garis menggunakan library matplotlib dengan menggunakan tensor pytorch. hasil output menunjukkan adanya plot garis yang naik yang menunjukkan nilai nilai dalam tensor A pada sumbu y. sementara sumbu x mempresentasikan indeks dari tensor

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

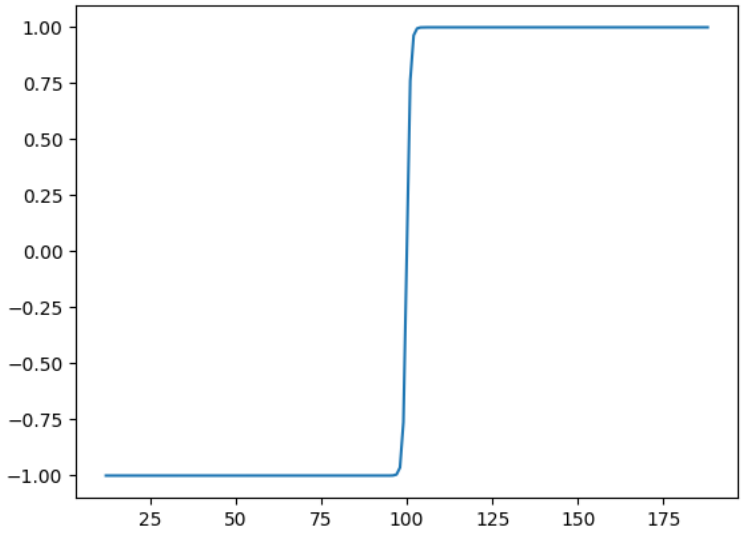
```
[<matplotlib.lines.Line2D at 0x7ae7efc119f0>]
```



hasil output tersebut merupakan plot garis yang menunjukkan nilai-nilai fungsi tangen hiperbolik dari tensor A pada sumbu y. dapat dilihat fungsi tangen memetakan nilai nilai tensor ke dalam rentang (-1,1)

```
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))
```

```
[<matplotlib.lines.Line2D at 0x7ae7efe5f100>]
```

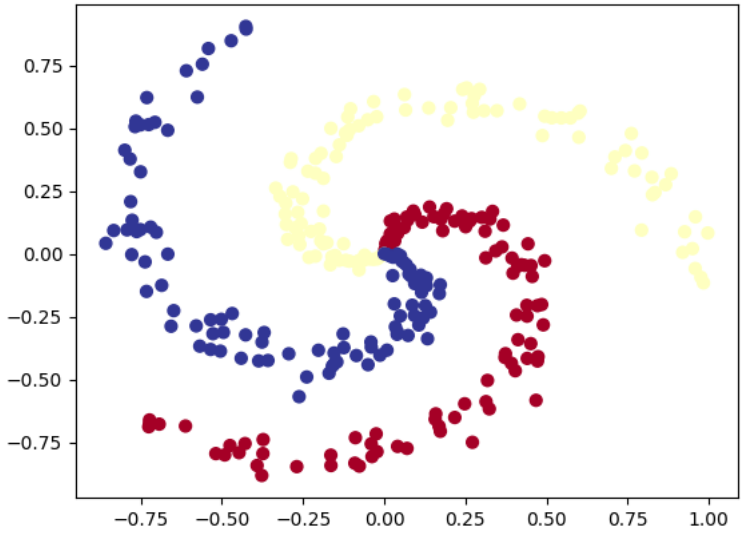


Hasil outputnya adalah plot garis yang menunjukkan nilai-nilai fungsi tangen hiperbolik manual dari tensor tensor_A pada sumbu y. Plot ini seharusnya serupa dengan hasil yang diperoleh menggunakan fungsi tangen hiperbolik bawaan PyTorch (torch.tanh), karena implementasi rumus yang digunakan dalam fungsi manual sesuai dengan rumus standar fungsi tangen hiperbolik.

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

- Split the data into training and test sets (80% train, 20% test) as well as turn it into PyTorch tensors.
- Construct a model capable of fitting the data (you may need a combination of linear and non-linear layers).
- Build a loss function and optimizer capable of handling multi-class data (optional extension: use the Adam optimizer instead of SGD, you may have to experiment with different values of the learning rate to get it working).
- Make a training and testing loop for the multi-class data and train a model on it to reach over 95% testing accuracy (you can use any accuracy measuring function here that you like).
- Plot the decision boundaries on the spirals dataset from your model predictions, the `plot_decision_boundary()` function should work for this dataset too.

```
# Code for creating a spiral dataset from CS231n
import numpy as np
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
N = 100 # number of points per class
D = 2 # dimensionality
K = 3 # number of classes
X = np.zeros((N*K,D)) # data matrix (each row = single example)
y = np.zeros(N*K, dtype='uint8') # class labels
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  X[ix] = np.c_[r*np.sin(t), r*np.cos(t)]
  y[ix] = j
# lets visualize the data
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)
plt.show()
```



kode tersebut digunakan untuk membuat dataset berbentuk spiral dengan tiga kelas menggunakan pendekatan yang diambil dari kursus CS231n. output yang dihasilkan berupa scatter plot yang menunjukkan dataset berbentuk spiral dengan tiga kelas yang dibuat secara sintetis. Setiap kelas ditandai dengan warna yang berbeda

```
# 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
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=RANDOM_SEED)
len(X_train), len(X_test), len(y_train), len(y_test)
```

```
(240, 60, 240, 60)
```

kode tersebut digunakan untuk mengubah data dari format NumPy menjadi tensor PyTorch dan kemudian membagi dataset menjadi set pelatihan dan pengujian menggunakan library scikit-learn.Hasil outputnya adalah tuple yang berisi panjang set pelatihan dan pengujian untuk fitur (X_train, X_test) dan label (y_train, y_test). untuk X_train dan y_train bernilai 240 sementara X_test dan y_test bernilai 60

```
# 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
acc_fn = Accuracy(task="multiclass", num_classes=3).to(device) # send accuracy function to device
acc_fn
```

```
MulticlassAccuracy()
```

kode tersebut menginstal dan mengimpor library torchmetrics, dan kemudian membuat instance dari metrik akurasi (accuracy) yang disebut acc_fn. Dengan membuat instance metrik akurasi ini, dapat digunakan untuk mengukur performa model pada set pelatihan dan pengujian selama proses pelatihan dan evaluasi. Metrik akurasi ini akan menghitung akurasi prediksi model terhadap label sebenarnya pada data yang diberikan.

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

class SpiralModel(nn.Module):
  def __init__(self):
    super().__init__()
    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)
    self.relu = nn.ReLU()

  def forward(self, x):
    return self.linear3(self.relu(self.linear2(self.relu(self.linear1(x)))))

model_1 = SpiralModel().to(device)
model_1
```

```
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()
  )
```

Model ini adalah model sederhana dengan tiga layer linear dan fungsi aktivasi ReLU antara setiap layer, digunakan untuk mempelajari pola dalam dataset berbentuk spiral.

```
# 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))
```

```
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')
```

kode tersebut digunakan untuk mengonversi data pelatihan dan pengujian ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU) menggunakan tensor PyTorch dan mencetak tipe data dari masing-masing tensor. Selanjutnya, model yang belum dilatih dievaluasi pada sebagian kecil dari data pelatihan

```
# Setup loss function and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_1.parameters(),
                             lr=0.02)
```

kode tersebut digunakan untuk menetapkan fungsi loss dan optimizer untuk pelatihan model.

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

# Loop over data
for epoch in range(epochs):
  ## Training
  model_1.train()
  # 1. forward pass
  y_logits = model_1(X_train)
  y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1)

  # 2. calculate the loss
  loss = loss_fn(y_logits, y_train)
  acc = acc_fn(y_pred, y_train)

  # 3. optimizer zero grad
  optimizer.zero_grad()

  # 4. loss backwards
  loss.backward()

  # 5. optimizer step step step
  optimizer.step()

  ## Testing
  model_1.eval()
  with torch.inference_mode():
    # 1. Forward pass
    test_logits = model_1(X_test)
```

kode tersebut adalah loop pelatihan (training loop) untuk model. Loop ini melibatkan beberapa langkah, seperti forward pass, perhitungan loss, backward pass, dan update parameter model menggunakan optimizer. Loop pelatihan berjalan sebanyak epoch yang ditetapkan, dan selama iterasi setiap epoch, model akan memperbarui parameter berdasarkan data pelatihan

```
# Print out what's happening
```

```
# 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)
```