Syifa Wanda Isnaini

1103201248

Machine Learning

02. Pytorch CClassification Exercise

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

```
Thu Feb 10 00:20:37 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| 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 P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P0    26W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

Kode ini adalah perintah shell yang digunakan untuk menampilkan informasi tentang perangkat GPU yang sedang digunakan. Dalam hal ini, !nvidia-smi digunakan untuk menjalankan perintah sistem NVIDIA System Management Interface (nvidia-smi) yang biasanya digunakan untuk memantau dan mengelola informasi tentang GPU NVIDIA yang terpasang pada sistem.

Penjelasan dari kode tersebut:

!nvidia-smi ! adalah karakter yang digunakan untuk menjalankan perintah shell dari dalam notebook atau lingkungan Python. nvidia-smi adalah perintah yang memberikan informasi tentang penggunaan GPU, status, dan detail lainnya. Dengan menjalankan perintah ini, kita dapat melihat informasi tentang GPU yang tersedia, seperti penggunaan memori, suhu, dan utilitas GPU. Jadi, kode ini digunakan untuk memeriksa informasi tentang perangkat GPU yang tersedia pada sistem. Perintah ini berguna untuk memastikan bahwa GPU dapat digunakan dan untuk melihat informasi spesifik tentang GPU yang sedang digunakan.

```
# Import torch
import torch

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

```
'cuda'
```

import torch

Mengimpor pustaka PyTorch, yang merupakan pustaka machine learning yang populer. device = "cuda" if torch.cuda.is_available() else "cpu"

Menggunakan ekspresi kondisional (if-else) untuk menentukan perangkat (device) yang akan digunakan. Jika torch.cuda.is_available() mengembalikan True (artinya GPU tersedia), maka device diatur sebagai "cuda" (GPU). Jika tidak, device diatur sebagai "cpu" (CPU). device

Mencetak nilai dari device, yang akan menjadi string "cuda" jika GPU tersedia, dan "cpu" jika tidak. Kode ini berguna untuk membuat skrip yang bersifat agnostik terhadap perangkat, sehingga dapat berjalan baik pada CPU maupun GPU tanpa perlu mengubah secara manual konfigurasi perangkat.

```
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 pustaka scikit-learn untuk membuat dataset yang menggambarkan dua bulan (moons). from sklearn.datasets import make_moons

Mengimpor fungsi make_moons dari modul datasets dalam pustaka scikit-learn. Fungsi ini digunakan untuk membuat dataset yang menciptakan dua bulan (moons). NUM_SAMPLES = 1000

Mendefinisikan konstanta NUM_SAMPLES yang menentukan jumlah total sampel dalam dataset. RANDOM_SEED = 42

Mendefinisikan konstanta RANDOM_SEED yang menentukan nilai seed untuk menghasilkan dataset dengan cara yang dapat direproduksi. X, y = make_moons(n_samples=NUM_SAMPLES, noise=0.07, random_state=RANDOM_SEED)

Memanggil fungsi make_moons untuk membuat dataset dengan jumlah sampel sebanyak NUM_SAMPLES. Parameter noise=0.07 menentukan seberapa banyak noise (gangguan acak) yang akan ditambahkan ke dataset. Parameter random_state=RANDOM_SEED memastikan bahwa dataset yang dihasilkan dapat direproduksi dengan menggunakan seed yang sama. X[:10], y[:10]

Mencetak 10 sampel pertama dari fitur (X) dan label (y) dataset untuk memberikan gambaran awal tentang struktur dataset.

Hasilnya akan menunjukkan 10 sampel pertama dari fitur dan label dataset yang telah dibuat menggunakan fungsi make_moons.

```
# 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

Kode ini mengubah data dari array NumPy ke dalam format DataFrame menggunakan pustaka Pandas. import pandas as pd

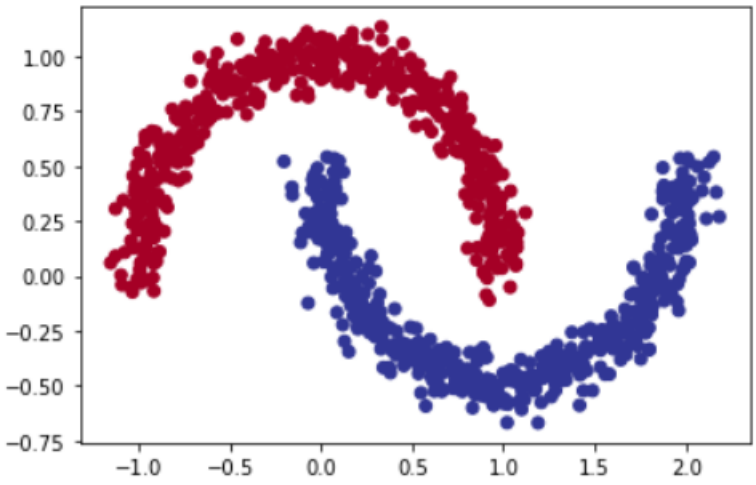
Mengimpor pustaka Pandas dengan alias pd, yang sering digunakan untuk manipulasi dan analisis data tabular. data_df = pd.DataFrame({"X0": X[:, 0], "X1": X[:, 1], "y": y})

Membuat DataFrame (data_df) dari array NumPy X dan y. Setiap kolom DataFrame diberi nama "X0", "X1", dan "y". Kolom "X0" dan "X1" mengambil nilai dari kolom pertama dan kedua dari array X, masing-masing. Kolom "y" mengambil nilai dari array y. data_df.head()

Menggunakan metode head() untuk menampilkan lima baris pertama dari DataFrame data_df. Ini memberikan gambaran awal tentang struktur dan konten DataFrame.

Hasilnya adalah DataFrame yang memiliki kolom "X0" dan "X1" untuk fitur serta kolom "y" untuk label. Ini memungki

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



Kode ini menggunakan pustaka Matplotlib untuk membuat scatter plot yang memvisualisasikan data yang telah dibuat. import matplotlib.pyplot as plt

Mengimpor pustaka Matplotlib dengan alias plt, yang sering digunakan untuk membuat visualisasi grafik. plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu)

Membuat scatter plot dari data. X[:, 0] dan X[:, 1] digunakan sebagai koordinat x dan y, masing-masing. c=y digunakan untuk memberikan warna pada setiap titik berdasarkan nilai dari array y. cmap=plt.cm.RdYlBu digunakan untuk memilih peta warna (color map) yang akan digunakan dalam plot. plt.show()

Menampilkan plot yang telah dibuat.

Hasilnya adalah scatter plot di mana setiap titik direpresentasikan oleh dua fitur (X[:, 0] dan X[:, 1]) dan diwarnai berdasarkan nilai dari array y. Plot ini memberikan gambaran visual tentang bagaimana dataset dua bulan (moons) tersebut terbagi menjadi dua kelas.

```
# 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)
```

X = torch.tensor(X, dtype=torch.float) dan y = torch.tensor(y, dtype=torch.float)

Mengubah array NumPy X dan y menjadi Tensors PyTorch. Tensors adalah struktur data fundamental dalam PyTorch yang dapat digunakan untuk menyimpan dan memanipulasi data numerik. from sklearn.model_selection import train_test_split

Mengimpor fungsi train_test_split dari modul model_selection dalam pustaka scikit-learn. Fungsi ini digunakan untuk membagi dataset menjadi dua bagian: satu untuk pelatihan (train) dan satu untuk pengujian (test). X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=RANDOM_SEED)

Memanggil fungsi train_test_split untuk membagi Tensors X dan y menjadi empat bagian: X_train (fitur latihan), X_test (fitur uji), y_train (label latihan), dan y_test (label uji). Parameter test_size=0.2 menentukan bahwa 20% dari data akan digunakan sebagai set pengujian. Parameter random_state=RANDOM_SEED memastikan bahwa pembagian data dapat direproduksi dengan menggunakan seed yang sama. len(X_train), len(X_test), len(y_train), len(y_test)

Mencetak panjang (jumlah sampel) dari masing-masing set data latihan dan uji.

Hasilnya adalah empat Tensor yang mewakili set data latihan dan uji, serta panjang (jumlah sampel) dari masing-masing set. Data ini siap digunakan untuk pelatihan dan evaluasi model.

```
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()
)
```

class MoonModelV0(nn.Module):

Mendefinisikan kelas MoonModelV0 yang merupakan turunan dari kelas nn.Module di PyTorch. Ini menandakan bahwa kelas ini akan digunakan untuk membuat model jaringan saraf.
def **init**(self, in_features, out_features, hidden_units):

Konstruktor kelas yang menginisialisasi model dengan lapisan-lapisan dan fungsi aktivasi yang diperlukan. in_features: Jumlah fitur masukan (input) ke model. out_features: Jumlah fitur keluaran (output) dari model. hidden_units: Jumlah unit pada lapisan tersembunyi. self.layer1 = nn.Linear(in_features=in_features, out_features=hidden_units)

Lapisan linear pertama dengan jumlah fitur masukan in_features dan jumlah fitur keluaran hidden_units. self.layer2 = nn.Linear(in_features=hidden_units, out_features=hidden_units)

Lapisan linear kedua dengan jumlah fitur masukan dan keluaran hidden_units. self.layer3 = nn.Linear(in_features=hidden_units, out_features=out_features)

Lapisan linear ketiga dengan jumlah fitur masukan hidden_units dan jumlah fitur keluaran out_features. self.relu = nn.ReLU()

Fungsi aktivasi ReLU (Rectified Linear Unit) yang diterapkan di antara setiap lapisan. def forward(self, x):

Metode forward yang mendefinisikan operasi perpindahan ke depan (forward pass) dari model. Fungsi ini menggambarkan cara data mengalir melalui lapisan-lapisan dan fungsi aktivasi model. return self.layer3(self.relu(self.layer2(self.relu(self.layer1(x))))))

Melakukan forward pass dengan mengalirkan input x melalui lapisan-lapisan dan fungsi aktivasi. model_0 = MoonModelV0(in_features=2, out_features=1, hidden_units=10).to(device)

Membuat instance dari model MoonModelV0 dengan jumlah fitur masukan 2, jumlah fitur keluaran 1, dan jumlah unit tersembunyi 10. Model ini kemudian dipindahkan ke perangkat target yang telah ditentukan sebelumnya (device), misalnya GPU jika tersedia.

```
model_0.state_dict()
```

```
OrderedDict([('layer1.weight', tensor([[ 0.5406,  0.5869],
                      [-0.1657,  0.6496],
                      [-0.1549,  0.1427],
                      [-0.3443,  0.4153],
                      [ 0.6233, -0.5188],
                      [ 0.6146,  0.1323],
                      [ 0.5224,  0.0958],
                      [ 0.3410, -0.0998],
                      [ 0.5451,  0.1045],
                      [-0.3301,  0.1802]], device='cuda:0')),
             ('layer1.bias',
              tensor([-0.3258, -0.0829, -0.2872,  0.4691, -0.5582, -0.3260, -0.1997, -0.4252,
                       0.0667, -0.6984], device='cuda:0')),
             ('layer2.weight',
              tensor([[ 0.2856, -0.2686,  0.2441,  0.0526, -0.1027,  0.1954,  0.0493,  0.2555,
                        0.0346, -0.0997],
                      [ 0.0850, -0.0858,  0.1331,  0.2823,  0.1828, -0.1382,  0.1825,  0.0566,
                        0.1606, -0.1927],
                      [-0.3130, -0.1222, -0.2426,  0.2595,  0.0911,  0.1310,  0.1000, -0.0055,
                        0.2475, -0.2247],
                      [ 0.0199, -0.2158,  0.0975, -0.1089,  0.0969, -0.0659,  0.2623, -0.1874,
                       -0.1886, -0.1886],
                      [ 0.2844,  0.1054,  0.3043, -0.2610, -0.3137, -0.2474, -0.2127,  0.1281,
                        0.1132,  0.2628],
                      [-0.1633, -0.2156,  0.1678, -0.1278,  0.1919, -0.0750,  0.1809, -0.2457,
                       -0.1596,  0.0964],
                      [ 0.0669, -0.0806,  0.1885,  0.2150, -0.2293, -0.1688,  0.2896, -0.1067,
                       -0.1121, -0.3060],
                      [-0.1811,  0.0790, -0.0417, -0.2295,  0.0074, -0.2160, -0.2683, -0.1741,
                       -0.2768, -0.2014],
                      [ 0.3161,  0.0597,  0.0974, -0.2949, -0.2077, -0.1053,  0.0494, -0.2783,
                       -0.1363, -0.1893],
                      [ 0.0009, -0.1177, -0.0219, -0.2143, -0.2171, -0.1845, -0.1082, -0.2496,
                        0.2651, -0.0628]], device='cuda:0')),
```

```
          0.2651, -0.0628]], device='cuda:0')),
('layer2.bias',
 tensor([ 0.2721,  0.0985, -0.2678,  0.2188, -0.0870, -0.1212, -0.2625, -0.3144,
          0.0905, -0.0691], device='cuda:0')),
('layer3.weight',
 tensor([[ 0.1231, -0.2595,  0.2348, -0.2321, -0.0546,  0.0661,  0.1633,  0.2553,
          0.2881, -0.2507]], device='cuda:0')),
('layer3.bias', tensor([0.0796], device='cuda:0'))])
```

model_0.state_dict() mengembalikan kamus (dictionary) yang berisi parameter atau bobot dari model model_0. Dalam konteks model PyTorch, state_dict adalah struktur data yang menyimpan semua parameter dari model, seperti bobot dan bias pada setiap lapisan. Berikut adalah potongan contoh output dari model_0.state_dict():

layer1.weight dan layer1.bias: Bobot dan bias dari lapisan pertama (layer1). layer2.weight dan layer2.bias: Bobot dan bias dari lapisan kedua (layer2). layer3.weight dan layer3.bias: Bobot dan bias dari lapisan ketiga (layer3). Struktur kamus ini memungkinkan kita untuk menyimpan dan memuat parameter model dengan mudah saat kita menyimpan atau meload model. Contoh penggunaannya adalah ketika kita ingin menyimpan model ke file atau memuat model dari file untuk penggunaan selanjutnya.

```
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 t
                            lr=0.1) # learning rate
```

Pada blok kode tersebut, Anda mendefinisikan fungsi kerugian (loss function) dan optimizer untuk model model_0 yang telah Anda buat sebelumnya.

Loss Function (nn.BCEWithLogitsLoss()):

nn.BCEWithLogitsLoss() digunakan ketika Anda memiliki layer sigmoid built-in di dalam model. Ini merupakan fungsi kerugian yang umum digunakan untuk tugas biner (binary classification). Layer sigmoid built-in membuat output model berupa logit yang belum melalui aktivasi sigmoid. Fungsi ini menggabungkan langkah sigmoid dan perhitungan loss (entropy log biner) menjadi satu, yang dapat meningkatkan stabilitas numerik dan kecepatan komputasi. Optimizer (torch.optim.SGD):

torch.optim.SGD adalah optimizer stokastik gradien turun (Stochastic Gradient Descent). Optimizer ini digunakan untuk menyesuaikan bobot model berdasarkan gradien loss terhadap bobot. params=model_0.parameters() menunjukkan bahwa kita ingin mengoptimalkan parameter atau bobot dari model model_0. lr=0.1 adalah learning rate, yaitu seberapa besar langkah yang diambil oleh optimizer untuk memperbarui bobot model pada setiap iterasi. Dengan menggabungkan fungsi kerugian dan optimizer ini, Anda dapat melatih model model_0 untuk tugas klasifikasi biner.

```
# 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([0.0019, 0.0094, 0.0161, 0.0185, 0.0284, 0.0192, 0.0291, 0.0196, 0.0258,
        0.0079], device='cuda:0', grad_fn=<SqueezeBackward0>)
Pred probs:
tensor([0.5005, 0.5024, 0.5040, 0.5046, 0.5071, 0.5048, 0.5073, 0.5049, 0.5065,
        0.5020], 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>)
```

Pada blok kode tersebut, Anda mengevaluasi beberapa output dari model model_0 pada 10 sampel pertama dari data latih (X_train). Berikut penjelasan setiap output:

Logits:

Logit adalah hasil keluaran model sebelum melewati fungsi aktivasi sigmoid. Dalam konteks tugas klasifikasi biner, logit mewakili seberapa yakin model terhadap setiap kelas. Prediction Probabilities:

Setelah logit diumpankan melalui fungsi sigmoid, kita mendapatkan nilai probabilitas untuk kelas positif. Fungsi sigmoid mengonversi logit menjadi rentang antara 0 dan 1, yang dapat diartikan sebagai probabilitas. Prediction Labels:

Label prediksi didapatkan dengan membulatkan nilai probabilitas. Pada tugas klasifikasi biner, jika probabilitas lebih besar dari 0.5, maka label prediksi adalah 1; sebaliknya, jika probabilitas kurang dari atau sama dengan 0.5, maka label prediksi adalah 0. Dengan mengevaluasi output-output tersebut, Anda dapat memahami lebih baik bagaimana model Anda merespon terhadap data latih dan melihat seberapa yakin model dalam memberikan prediksi untuk setiap sampel.

```
# 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 f
acc_fn
```

```
MulticlassAccuracy()
```

Pada blok kode tersebut, Anda menginstall library torchmetrics dan menggunakan metrik Accuracy untuk menghitung akurasi dari model. Berikut beberapa penjelasan:

!pip -q install torchmetrics

Perintah ini digunakan untuk menginstall library torchmetrics secara diam-diam (-q). from torchmetrics import Accuracy

Anda mengimport metrik Accuracy dari library torchmetrics. acc_fn = Accuracy(task="multiclass", num_classes=2).to(device)

Anda membuat instance dari metrik Accuracy dengan parameter: task="multiclass": Menunjukkan bahwa kita sedang melakukan klasifikasi multikelas. num_classes=2: Menunjukkan bahwa terdapat dua kelas (klasifikasi biner). to(device): Menyimpan instance metrik di perangkat yang sama dengan perangkat yang digunakan untuk model (device). Metrik akurasi (Accuracy) ini dapat digunakan untuk mengukur sejauh mana model kita berhasil dalam memprediksi kelas yang benar. Dengan menggunakan metrik ini, Anda dapat dengan mudah mengukur dan memantau kinerja model pada setiap iterasi atau epoch selama pelatihan.

```
# 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)
```

```
# 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())
```

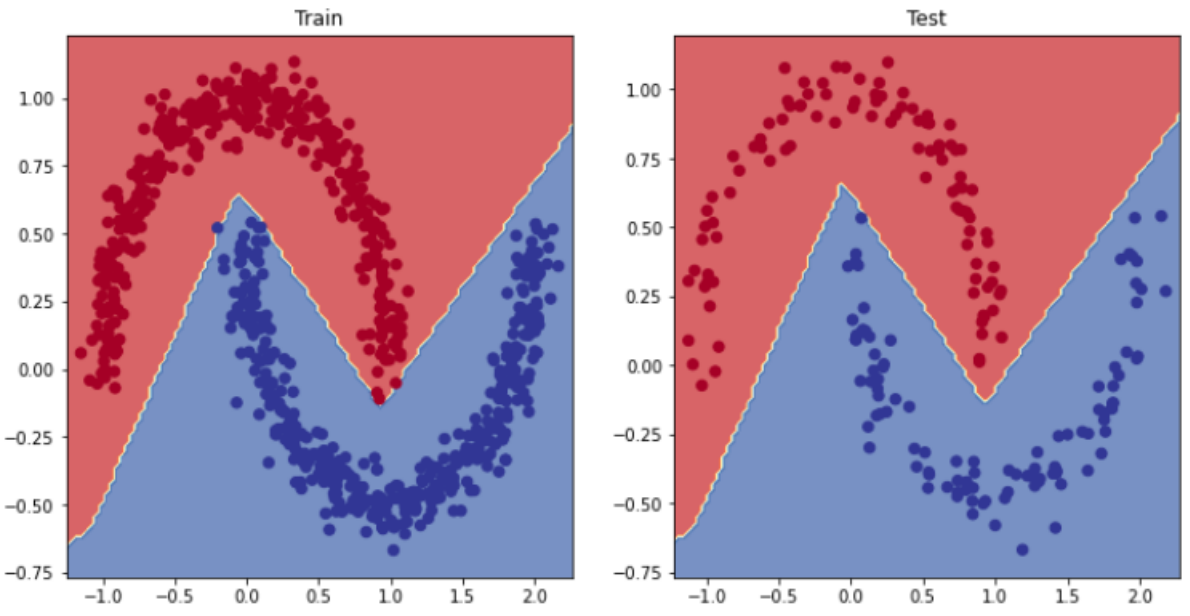
Blok kode di atas mendefinisikan fungsi plot_decision_boundary, yang digunakan untuk menggambar batas keputusan (decision boundary) dari model pada data dengan dua fitur. Berikut penjelasan fungsinya:

Input:

model: Model PyTorch yang akan digunakan untuk menggambar batas keputusan. X: Data fitur yang akan digunakan untuk menggambar batas keputusan. y: Label yang sesuai dengan data fitur. Langkah-langkah:

Fungsi mengubah model dan data ke CPU karena lebih efisien untuk digunakan bersama-sama dengan NumPy dan Matplotlib. Membuat grid fitur (xx, yy) untuk digunakan dalam plot batas keputusan. Mengonversi grid fitur ke tensor PyTorch untuk digunakan dalam model. Menggunakan model untuk membuat prediksi pada setiap titik pada grid fitur. Jika data memiliki lebih dari dua kelas (multi-class), fungsi menggunakan softmax untuk mendapatkan label prediksi. Jika data hanya memiliki dua kelas (binary), fungsi menggunakan fungsi sigmoid dan pembulatan untuk mendapatkan label prediksi. Mereshape label prediksi untuk sesuai dengan grid dan menggambar kontur batas keputusan dengan menggunakan plt.contourf. Membuat scatter plot untuk menunjukkan data fitur dengan warna sesuai dengan label sesungguhnya (y). Menyesuaikan batas sumbu x dan y sesuai dengan batas grid. Fungsi ini membantu memberikan visualisasi tentang seberapa baik model dapat memisahkan kelas-kelas berbeda pada data fitur dua dimensi.

```
# 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)
```



Blok kode di atas menggunakan fungsi plot_decision_boundary untuk menggambar batas keputusan pada dua set data berbeda: set pelatihan (X_train, y_train) dan set pengujian (X_test, y_test). Berikut adalah penjelasan untuk blok kode tersebut:

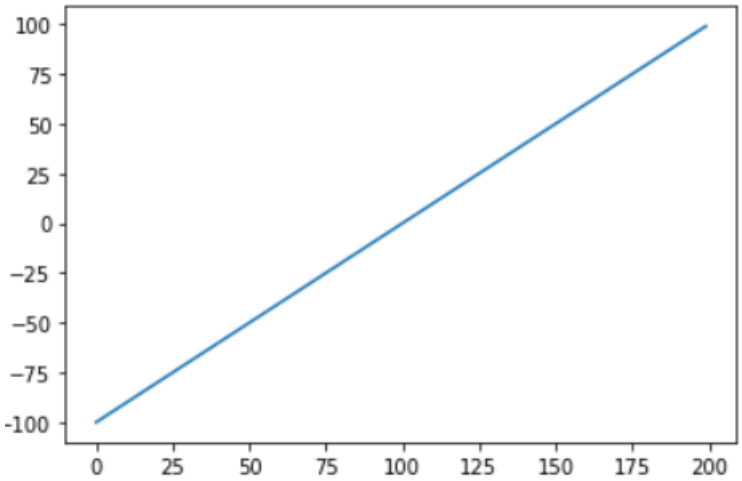
Input:

model_0: Model PyTorch yang ingin dilihat batas keputusannya. X_train, y_train: Data pelatihan dan label yang akan digunakan untuk menggambar batas keputusan pada set pelatihan. X_test, y_test: Data pengujian dan label yang akan digunakan untuk menggambar batas keputusan pada set pengujian. Langkah-langkah:

Membuat gambar berukuran (12, 6) dengan dua subplot. Pada subplot pertama: Menetapkan judul "Train". Memanggil plot_decision_boundary untuk menggambar batas keputusan pada data pelatihan. Pada subplot kedua: Menetapkan judul "Test". Memanggil plot_decision_boundary untuk menggambar batas keputusan pada data pengujian. Blok kode ini memberikan pemahaman visual tentang seberapa baik model dapat memisahkan kelas pada kedua set data: pelatihan dan pengujian.

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

```
[<matplotlib.lines.Line2D at 0x7f37332d5c10>]
```



Blok kode di atas membuat tensor tensor_A yang berisi nilai dari -100 hingga 99 (total 200 elemen) dengan interval 1, dan kemudian memplotnya menggunakan plt.plot(). Ini akan menghasilkan grafik garis yang menggambarkan nilai tensor terhadap indeksnya. Berikut adalah penjelasan singkat:

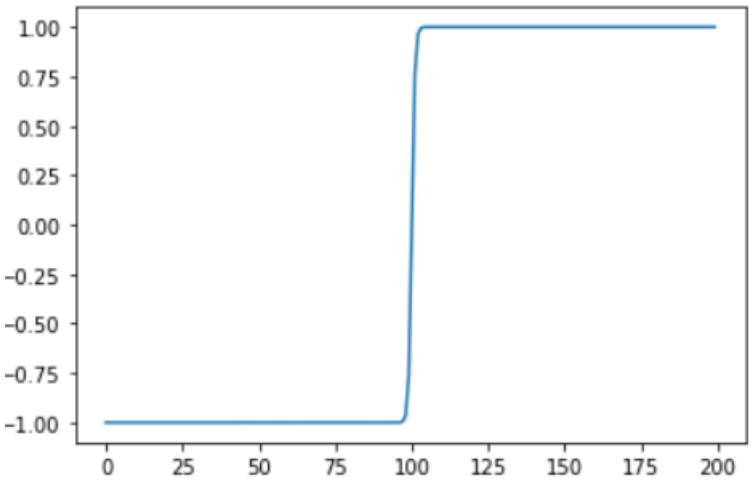
torch.arange(-100, 100, 1): Membuat tensor PyTorch yang berisi nilai dari -100 hingga 99 dengan interval 1.

plt.plot(tensor_A): Menggunakan Matplotlib untuk membuat plot garis dari nilai-nilai dalam tensor. Indeks tensor akan digunakan sebagai sumbu x, sementara nilai tensor akan digunakan sebagai sumbu y.

Grafik yang dihasilkan akan menunjukkan pola linear, di mana sumbu x mewakili indeks elemen dalam tensor, dan sumbu y mewakili nilai elemen tensor.

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

```
[<matplotlib.lines.Line2D at 0x7f3733254b10>]
```



Blok kode tersebut menggunakan fungsi hiperbolik tangen (torch.tanh()) pada tensor tensor_A dan memplot hasilnya menggunakan plt.plot(). Fungsi hiperbolik tangen adalah fungsi aktivasi yang umum digunakan dalam jaringan saraf. Berikut adalah penjelasan singkat:

torch.tanh(tensor_A): Menghitung nilai tangen hiperbolik dari setiap elemen dalam tensor tensor_A.

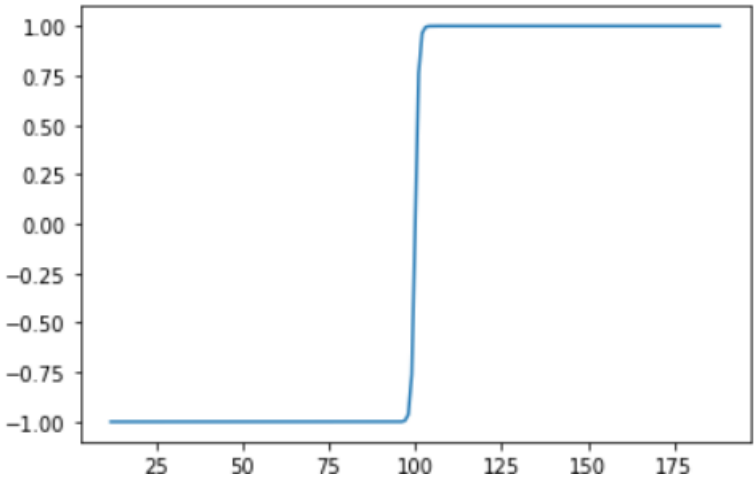
plt.plot(torch.tanh(tensor_A)): Memplot nilai-nilai tangen hiperbolik terhadap indeks tensor menggunakan Matplotlib.

Grafik yang dihasilkan akan menunjukkan kurva fungsi tangen hiperbolik, yang memiliki rentang nilai antara -1 dan 1.

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

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

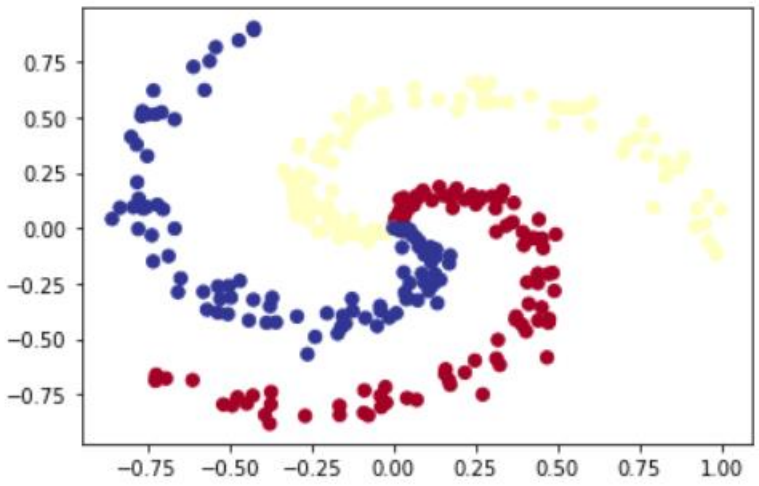
```
[<matplotlib.lines.Line2D at 0x7f37331d55d0>]
```



Fungsi yang didefinisikan, tanh(x), mengimplementasikan rumus matematika dari fungsi hiperbolik tangen (tanh). Fungsi ini menggunakan operasi elemen-wise pada tensor x. Berikut adalah beberapa poin penjelasan:

torch.exp(x): Menghitung eksponensial dari setiap elemen dalam tensor x. torch.exp(-x): Menghitung eksponensial dari setiap elemen dalam tensor -x. torch.exp(x) + torch.exp(-x): Menjumlahkan hasil eksponensial dari x dan -x. (torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-x)): Menghitung nilai tangen hiperbolik menggunakan rumus yang diberikan. Selanjutnya, hasil dari tanh(tensor_A) diproses dan diplot menggunakan plt.plot() seperti sebelumnya, menghasilkan kurva fungsi hiperbolik tangen pada rentang nilai tensor.

```
# 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 yang diberikan menghasilkan dataset sintetis yang dikenal sebagai "dataset spiral". Dataset ini terdiri dari tiga lengan spiral dalam ruang 2D, di mana setiap lengan sesuai dengan kelas yang berbeda. Berikut adalah penjelasan singkat dari kode tersebut:

N: Jumlah titik per kelas. D: Dimensi dataset (2D dalam kasus ini). K: Jumlah kelas (3 dalam kasus ini). Pada loop for, dibuat titik data untuk setiap kelas, menciptakan pola spiral. Posisi setiap titik ditentukan oleh koordinat polar (r, t) yang diubah menjadi koordinat Kartesian (r * sin(t), r * cos(t)). Variabel r mewakili radius, dan t mewakili sudut. Beberapa noise acak ditambahkan pada sudut (t) untuk memperkenalkan variasi.

Dataset yang dihasilkan divisualisasikan menggunakan plt.scatter(), di mana titik-titik dari setiap kelas diwarnai secara berbeda.

Dataset seperti ini sering digunakan untuk menguji dan mengilustrasikan perilaku algoritma pembelajaran mesin, khususnya algoritma yang dirancang untuk tugas klasifikasi non-linear.

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

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

Kode tersebut mengubah data dari Numpy arrays menjadi PyTorch tensors. Langkah ini umumnya diperlukan karena model PyTorch biasanya bekerja dengan tensors. Berikut adalah penjelasan singkat:

X = torch.from_numpy(X).type(torch.float): Mengubah array Numpy X menjadi tensor PyTorch dengan tipe data float32. Ini penting karena kebanyakan model PyTorch mengharapkan input dalam format tensor float32.

y = torch.from_numpy(y).type(torch.LongTensor): Mengubah array Numpy y menjadi tensor PyTorch dengan tipe data LongTensor. Pada tugas klasifikasi, label sering kali perlu diwakili oleh tensor dengan tipe data LongTensor.

train_test_split: Memisahkan data menjadi set pelatihan dan pengujian. Proporsi data yang digunakan untuk pengujian adalah 20%.

len(X_train), len(X_test), len(y_train), len(y_test): Menampilkan panjang dari setiap bagian data (pelatihan dan pengujian).

Dengan langkah-langkah ini, data telah disiapkan untuk digunakan dalam pelatihan dan pengujian model PyTorch.

```
# 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 f
acc_fn
```

```
MulticlassAccuracy()
```

Kode tersebut menginstal library torchmetrics dan membuat instance dari metrik akurasi untuk digunakan selama pelatihan dan evaluasi model. Berikut adalah penjelasan singkat:

!pip -q install torchmetrics: Instalasi torchmetrics menggunakan perintah shell di notebook Colab. Library ini menyediakan berbagai metrik evaluasi yang dapat digunakan untuk melacak performa model.

from torchmetrics import Accuracy: Mengimpor metrik akurasi dari library torchmetrics. Dalam hal ini, kita menggunakan metrik akurasi untuk tugas klasifikasi multi-kelas.

acc_fn = Accuracy(task="multiclass", num_classes=3).to(device): Membuat instance dari metrik akurasi dengan parameter task="multiclass" untuk menunjukkan bahwa kita bekerja pada tugas klasifikasi multi-kelas, dan num_classes=3 untuk menentukan jumlah kelas yang diharapkan oleh metrik. Instance tersebut juga dipindahkan ke perangkat yang sesuai (CPU atau GPU) menggunakan metode .to(device).

Metrik akurasi ini akan digunakan untuk mengevaluasi performa model selama dan setelah pelatihan.

```
# 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()
)
```

Kode di atas mempersiapkan model SpiralModel untuk tugas klasifikasi pada dataset spiral. Berikut adalah penjelasan singkatnya:

device = "cuda" if torch.cuda.is_available() else "cpu": Menentukan perangkat (device) yang akan digunakan untuk pelatihan model. Jika GPU tersedia, maka cuda (GPU) akan digunakan; jika tidak, maka cpu (CPU) akan digunakan.

class SpiralModel(nn.Module):: Mendefinisikan kelas model SpiralModel yang merupakan turunan dari kelas nn.Module dari PyTorch.

def **init**(self):: Konstruktor untuk inisialisasi model. Di dalamnya, terdapat definisi layer-layer yang akan digunakan oleh model.

self.linear1 = nn.Linear(in_features=2, out_features=10): Layer linear pertama dengan input features sebanyak 2 dan output features sebanyak 10.

self.linear2 = nn.Linear(in_features=10, out_features=10): Layer linear kedua dengan input features sebanyak 10 dan output features sebanyak 10.

self.linear3 = nn.Linear(in_features=10, out_features=3): Layer linear ketiga dengan input features sebanyak 10 dan output features sebanyak 3 (sesuai dengan jumlah kelas pada dataset spiral).

self.relu = nn.ReLU(): Fungsi aktivasi ReLU yang akan diaplikasikan setelah setiap layer linear.

def forward(self, x):: Metode forward yang mendefinisikan komputasi maju (forward pass) dari model. Data input x melewati serangkaian layer dan fungsi aktivasi untuk menghasilkan output.

model_1 = SpiralModel().to(device): Membuat instance dari model SpiralModel dan memindahkannya ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

Dengan ini, model model_1 siap untuk pelatihan dan evaluasi pada dataset 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 di atas mempersiapkan data dan mengevaluasi model sebelum dilakukan pelatihan. Berikut adalah penjelasan singkatnya:

X_train, y_train = X_train.to(device), y_train.to(device): Memindahkan data latih ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

print(X_train.dtype, X_test.dtype, y_train.dtype, y_test.dtype): Menampilkan tipe data (dtype) dari tensor-tensor data. Outputnya akan menunjukkan apakah data telah berada di GPU atau CPU.

print("Logits:"), print("Pred probs:"), print("Pred labels:"): Menampilkan output model untuk 10 data pertama pada data latih.

"Logits:": Menampilkan nilai raw output model sebelum diaplikasikan fungsi softmax.

"Pred probs:": Menampilkan nilai probabilitas prediksi untuk setiap kelas setelah diaplikasikan fungsi softmax.

"Pred labels:": Menampilkan label prediksi yang dihasilkan dengan mengambil argmax dari nilai softmax.

Dengan melakukan langkah-langkah ini sebelum pelatihan, Anda dapat melihat output model sebelum mempelajari pola-pola dalam data. Setelah model dilatih, output ini akan berubah sesuai dengan kemampuan model untuk memahami dan memprediksi data.

```
# 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 f
acc_fn
```

```
MulticlassAccuracy()
```

Kode tersebut menginstal library torchmetrics dan membuat instance dari metrik akurasi untuk digunakan selama pelatihan dan evaluasi model. Berikut adalah penjelasan singkat:

!pip -q install torchmetrics: Instalasi torchmetrics menggunakan perintah shell di notebook Colab. Library ini menyediakan berbagai metrik evaluasi yang dapat digunakan untuk melacak performa model.

from torchmetrics import Accuracy: Mengimpor metrik akurasi dari library torchmetrics. Dalam hal ini, kita menggunakan metrik akurasi untuk tugas klasifikasi multi-kelas.

acc_fn = Accuracy(task="multiclass", num_classes=3).to(device): Membuat instance dari metrik akurasi dengan parameter task="multiclass" untuk menunjukkan bahwa kita bekerja pada tugas klasifikasi multi-kelas, dan num_classes=3 untuk menentukan jumlah kelas yang diharapkan oleh metrik. Instance tersebut juga dipindahkan ke perangkat yang sesuai (CPU atau GPU) menggunakan metode .to(device).

Metrik akurasi ini akan digunakan untuk mengevaluasi performa model selama dan setelah pelatihan.

```
# 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()
)
```


Kode di atas mempersiapkan model SpiralModel untuk tugas klasifikasi pada dataset spiral. Berikut adalah penjelasan singkatnya:

device = "cuda" if torch.cuda.is_available() else "cpu": Menentukan perangkat (device) yang akan digunakan untuk pelatihan model. Jika GPU tersedia, maka cuda (GPU) akan digunakan; jika tidak, maka cpu (CPU) akan digunakan.

class SpiralModel(nn.Module):: Mendefinisikan kelas model SpiralModel yang merupakan turunan dari kelas nn.Module dari PyTorch.

def **init**(self):: Konstruktor untuk inisialisasi model. Di dalamnya, terdapat definisi layer-layer yang akan digunakan oleh model.

self.linear1 = nn.Linear(in_features=2, out_features=10): Layer linear pertama dengan input features sebanyak 2 dan output features sebanyak 10.

self.linear2 = nn.Linear(in_features=10, out_features=10): Layer linear kedua dengan input features sebanyak 10 dan output features sebanyak 10.

self.linear3 = nn.Linear(in_features=10, out_features=3): Layer linear ketiga dengan input features sebanyak 10 dan output features sebanyak 3 (sesuai dengan jumlah kelas pada dataset spiral).

self.relu = nn.ReLU(): Fungsi aktivasi ReLU yang akan diaplikasikan setelah setiap layer linear.

def forward(self, x):: Metode forward yang mendefinisikan komputasi maju (forward pass) dari model. Data input x melewati serangkaian layer dan fungsi aktivasi untuk menghasilkan output.

model_1 = SpiralModel().to(device): Membuat instance dari model SpiralModel dan memindahkannya ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

Dengan ini, model model_1 siap untuk pelatihan dan evaluasi pada dataset 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 di atas mempersiapkan data dan mengevaluasi model sebelum dilakukan pelatihan. Berikut adalah penjelasan singkatnya:

X_train, y_train = X_train.to(device), y_train.to(device): Memindahkan data latih ke perangkat yang telah ditentukan sebelumnya (GPU atau CPU).

print(X_train.dtype, X_test.dtype, y_train.dtype, y_test.dtype): Menampilkan tipe data (dtype) dari tensor-tensor data. Outputnya akan menunjukkan apakah data telah berada di GPU atau CPU.

print("Logits:"), print("Pred probs:"), print("Pred labels:"): Menampilkan output model untuk 10 data pertama pada data latih.

"Logits:": Menampilkan nilai raw output model sebelum diaplikasikan fungsi softmax.

"Pred probs:": Menampilkan nilai probabilitas prediksi untuk setiap kelas setelah diaplikasikan fungsi softmax.

"Pred labels:": Menampilkan label prediksi yang dihasilkan dengan mengambil argmax dari nilai softmax.

Dengan melakukan langkah-langkah ini sebelum pelatihan, Anda dapat melihat output model sebelum mempelajari pola-pola dalam data. Setelah model dilatih, output ini akan berubah sesuai dengan kemampuan model untuk memahami dan memprediksi data.

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

Kode di atas menyiapkan fungsi kerugian (loss function) dan pengoptimal (optimizer) untuk model model_1. Berikut adalah penjelasan singkatnya:

loss_fn = nn.CrossEntropyLoss(): Fungsi kerugian yang dipilih adalah CrossEntropyLoss, yang umumnya digunakan untuk masalah klasifikasi dengan beberapa kelas. Fungsi ini menggabungkan langkah-langkah softmax dan perhitungan kerugian log likelihood secara otomatis.

optimizer = torch.optim.Adam(model_1.parameters(), lr=0.02): Optimizer yang digunakan adalah Adam, yang merupakan algoritma optimasi yang populer. model_1.parameters() digunakan untuk menyertakan parameter-model yang akan dioptimalkan oleh Adam. Parameter ini melibatkan parameter-parameter dari lapisan-lapisan linear (nn.Linear) dalam model. lr=0.02 adalah laju pembelajaran (learning rate) yang digunakan oleh optimizer.

Dengan mengonfigurasi loss function dan optimizer ini, Anda dapat melatih model Anda menggunakan backpropagation untuk mengoptimalkan parameter dan mengurangi kerugian pada data pelatihan.

```python
# 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)
    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)
```

```python
    # 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: 1.12 Acc: 0.32 | Test loss: 0.91 Test acc: 0.37
Epoch: 100 | Loss: 0.45 Acc: 0.78 | Test loss: 0.32 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
```

Kode di atas adalah loop pelatihan (training loop) untuk model model_1. Berikut adalah penjelasan singkatnya:

for epoch in range(epochs):: Loop ini akan berjalan sebanyak epochs kali, di mana satu epoch adalah satu kali iterasi melalui seluruh data pelatihan.

model_1.train(): Metode train() digunakan untuk mengatur model ke mode pelatihan, yang diperlukan karena beberapa lapisan (layers), seperti dropout atau batch normalization, dapat berperilaku berbeda saat pelatihan dibandingkan dengan evaluasi.

y_logits = model_1(X_train): Menghasilkan logit (nilai sebelum softmax) dari model untuk data pelatihan.

y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1): Menerapkan softmax pada logit untuk mendapatkan distribusi probabilitas, dan kemudian mengambil kelas dengan probabilitas tertinggi sebagai prediksi.

loss = loss_fn(y_logits, y_train): Menghitung kerugian antara logit yang dihasilkan oleh model dan label sebenarnya.

optimizer.zero_grad(): Mengatur gradien parameter-model menjadi nol sebelum melakukan backpropagation.

loss.backward(): Menghitung gradien dari fungsi kerugian terhadap parameter-model menggunakan backpropagation.

optimizer.step(): Melakukan satu langkah optimasi menggunakan gradien yang dihitung pada langkah sebelumnya.

model_1.eval(): Metode eval() mengatur model ke mode evaluasi, yang berguna ketika kita ingin melakukan evaluasi pada data uji.

with torch.inference_mode():: Dengan menggunakan torch.inference_mode(), kita dapat memastikan bahwa model beroperasi dalam mode evaluasi dan tidak melacak gradien atau memperbarui parameter.

test_logits = model_1(X_test): Menghasilkan logit untuk data uji.

test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1): Menerapkan softmax pada logit untuk mendapatkan prediksi kelas pada data uji.

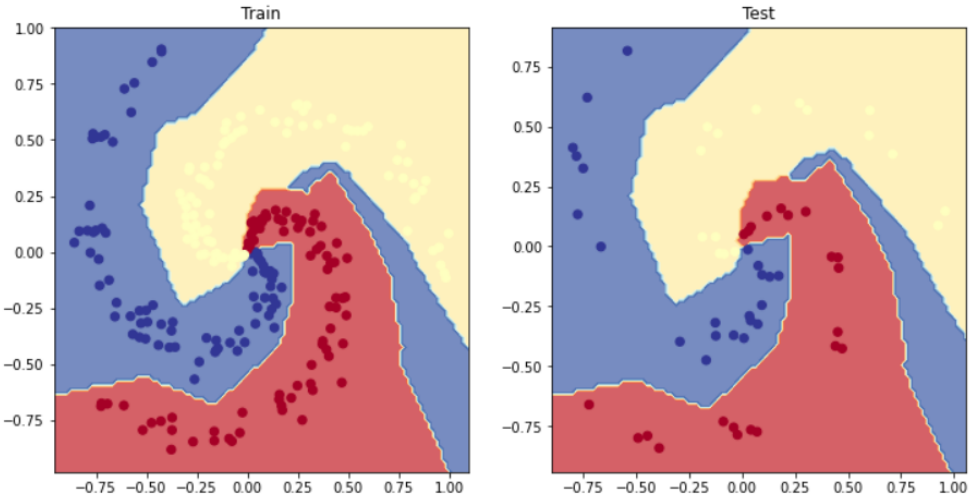
test_loss = loss_fn(test_logits, y_test): Menghitung kerugian pada data uji.

test_acc = acc_fn(test_pred, y_test): Menghitung akurasi pada data uji.

Jika epoch habis dibagi oleh 100, cetak informasi tentang loss dan akurasi pada data pelatihan dan uji.

Loop ini akan berulang sebanyak epochs kali untuk melatih model dan memantau kinerjanya pada setiap iterasi.

```
# 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)
```



Kode di atas digunakan untuk memvisualisasikan batas keputusan (decision boundaries) yang dihasilkan oleh model model_1 pada data pelatihan dan data uji. Berikut adalah penjelasan singkatnya:

plt.figure(figsize=(12, 6)): Membuat sebuah gambar (figure) dengan ukuran 12x6.

plt.subplot(1, 2, 1): Membuat subplot pertama dalam grid 1x2. Ini akan digunakan untuk memplot batas keputusan pada data pelatihan.

plt.title("Train"): Menambahkan judul untuk subplot pertama, menunjukkan bahwa ini adalah plot untuk data pelatihan.

plot_decision_boundary(model_1, X_train, y_train): Memanggil fungsi plot_decision_boundary untuk memplot batas keputusan pada data pelatihan menggunakan model model_1.

plt.subplot(1, 2, 2): Membuat subplot kedua dalam grid 1x2. Ini akan digunakan untuk memplot batas keputusan pada data uji.

plt.title("Test"): Menambahkan judul untuk subplot kedua, menunjukkan bahwa ini adalah plot untuk data uji.

plot_decision_boundary(model_1, X_test, y_test): Memanggil fungsi plot_decision_boundary untuk memplot batas keputusan pada data uji menggunakan model model_1.

Keseluruhan, ini memberikan visualisasi tentang sejauh mana model dapat memisahkan kelas-kelas dalam data pelatihan dan uji.