Nama : Rizka Rahmadina

NIM : 1103204115

Technical Report :

Reproduce_Codes_02_pytorch_classification

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

Kode !nvidia-smi adalah perintah shell yang digunakan untuk mengecek informasi tentang GPU (Graphics Processing Unit) pada sistem. Ini biasanya digunakan di lingkungan yang mendukung NVIDIA GPU, seperti ketika menggunakan sistem dengan kartu grafis NVIDIA. Kode ini memberikan informasi langsung tentang GPU yang terpasang pada sistem. Ini berguna untuk memastikan bahwa GPU diakses dengan benar atau untuk memonitor penggunaan GPU saat menjalankan tugas yang memerlukan daya komputasi grafis, seperti pelatihan model deep learning.

- ! digunakan di beberapa lingkungan seperti Jupyter Notebook atau IPython untuk menjalankan perintah shell langsung dari sel kode.
- nvidia-smi adalah perintah yang digunakan untuk memeriksa informasi tentang kartu grafis NVIDIA, seperti model, penggunaan GPU, suhu, dan sebagainya.

```
# Import torch
import torch

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

- import torch: Mengimpor library PyTorch ke dalam program.
- device = "cuda" if torch.cuda.is_available() else "cpu": Mengecek apakah GPU (CUDA) tersedia pada sistem. Jika CUDA tersedia, variabel device diatur menjadi string "cuda", menunjukkan penggunaan GPU. Jika tidak,

variabel device diatur menjadi string "cpu", menunjukkan penggunaan CPU sebagai alternatif.

- device: Menampilkan nilai dari variabel device, yang menunjukkan perangkat yang akan digunakan (GPU atau CPU) selama eksekusi kode.

Kode tersebut adalah bagian dari pengaturan awal untuk menggunakan library PyTorch di lingkungan yang mendukung GPU (CUDA), untuk menentukan perangkat (device) yang akan digunakan untuk eksekusi operasi PyTorch selanjutnya. Jika GPU tersedia, maka operasi PyTorch akan dijalankan di GPU, jika tidak, akan menggunakan CPU sebagai alternatif. Hal ini membantu dalam penulisan kode yang agnostik perangkat, sehingga kode dapat dijalankan dengan baik baik pada CPU maupun GPU tanpa perubahan besar dalam kode.

```
from sklearn.datasets import make_moons
import torch

NUM_SAMPLES = 1200  # Changed the number of samples
RANDOM_SEED = 33  # Changed the random seed

# Create binary classification dataset with make_moons
moon_data = make_moons(n_samples=NUM_SAMPLES, noise=0.09,
random_state=RANDOM_SEED)
X, y = moon_data

# Convert data to PyTorch tensors
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.float32).unsqueeze(1)  # Add an
extra dimension for compatibility

# Display the first 10 samples
print(f"First 10 X samples:\n{X_tensor[:10]}")
print(f"First 10 y samples:\n{y_tensor[:10]}")
```

- from sklearn.datasets import make_moons: Mengimpor fungsi make_moons dari modul sklearn.datasets, yang digunakan untuk membuat dataset dengan pola "moons" atau bulan sabit.
- import torch: Mengimpor library PyTorch.

- NUM_SAMPLES = 1200 dan RANDOM_SEED = 33: Menetapkan jumlah sampel (data points) dalam dataset (NUM_SAMPLES) dan seed untuk pengacakan (RANDOM_SEED).
- moon_data = make_moons(n_samples=NUM_SAMPLES, noise=0.09, random_state=RANDOM_SEED): Membuat dataset dengan pola "moons" sejumlah NUM_SAMPLES dengan tingkat kebisingan (noise) sebesar 0.09, dan menggunakan seed pengacakan sebesar RANDOM_SEED.
- X, y = moon_data: Memisahkan features (X) dan labels (y) dari dataset hasil fungsi make_moons.
- X_tensor = torch.tensor(X, dtype=torch.float32): Mengonversi features (X) ke dalam PyTorch tensor dengan tipe data float32.
- y_tensor = torch.tensor(y, dtype=torch.float32).unsqueeze(1): Mengonversi labels (y) ke dalam PyTorch tensor dengan tipe data float32, dan menambahkan dimensi ekstra menggunakan unsqueeze(1) untuk kompatibilitas dengan operasi PyTorch.
- print(f"First 10 X samples:\n{X_tensor[:10]}") dan print(f"First 10 y samples:\n{y_tensor[:10]}"): Menampilkan sepuluh sampel pertama dari features (X_tensor) dan labels (y_tensor), memberikan wawasan tentang struktur data yang telah dibuat.

Kode tersebut bertujuan untuk membuat dataset menggunakan fungsi make_moons dari scikit-learn dan mengonversinya ke dalam format yang kompatibel dengan PyTorch tensors, menciptakan dataset dengan pola "moons" menggunakan make_moons, dan mengonversi features dan labels ke dalam format tensor PyTorch untuk digunakan dalam pemodelan dengan PyTorch.

```
import pandas as pd
import numpy as np  # Import numpy for generating different numerical
values

NUM_SAMPLES_NEW = 1200  # Different number of samples
RANDOM_SEED_NEW = 55  # Different random seed

# Create binary classification dataset with make_moons
moon_data_new = make_moons(n_samples=NUM_SAMPLES_NEW, noise=0.1,
random_state=RANDOM_SEED_NEW)
X_new, y_new = moon_data_new
```

```
# Convert data to a DataFrame
data_df_new = pd.DataFrame({"Feature_0": X_new[:, 0],
                            "Feature_1": X_new[:, 1],
                            "Label": y_new})
data_df_new.head()
```

- import pandas as pd: Mengimpor library pandas dengan alias pd.
- import numpy as np: Mengimpor library numpy dengan alias np. Meskipun tidak digunakan secara langsung dalam kode ini, numpy sering digunakan untuk melakukan operasi numerik.
- NUM_SAMPLES_NEW = 1200 dan RANDOM_SEED_NEW = 55: Menetapkan jumlah sampel (data points) dalam dataset yang baru (NUM_SAMPLES_NEW) dan seed untuk pengacakan yang baru (RANDOM_SEED_NEW).
- moon_data_new = make_moons(n_samples=NUM_SAMPLES_NEW, noise=0.1, random_state=RANDOM_SEED_NEW): Membuat dataset baru dengan pola "moons" menggunakan jumlah sampel dan seed pengacakan yang baru.
- X_new, y_new = moon_data_new: Memisahkan features (X_new) dan labels (y_new) dari dataset baru.
- data_df_new = pd.DataFrame({"Feature_0": X_new[:, 0], "Feature_1": X_new[:, 1], "Label": y_new}): Membuat objek DataFrame data_df_new dengan kolom "Feature_0" dan "Feature_1" untuk features, serta kolom "Label" untuk labels. Data dari features dan labels diambil dari X_new dan y_new.
- data_df_new.head(): Menampilkan lima baris pertama dari DataFrame data_df_new menggunakan metode head().

Kode tersebut mirip dengan kode sebelumnya, tetapi kali ini hasil dataset dibuat dalam bentuk objek DataFrame menggunakan library pandas, menciptakan dataset baru dengan pola "moons" seperti sebelumnya, namun kali ini menyimpan hasilnya dalam bentuk DataFrame pandas. DataFrame ini dapat digunakan dengan mudah untuk analisis data lebih lanjut atau untuk mempersiapkan data untuk pemodelan machine learning dengan menggunakan library PyTorch atau scikit-learn.

```
import matplotlib.pyplot as plt
import numpy as np

# Visualize the data on a plot
plt.scatter(X_new[:, 0], X_new[:, 1], c=y_new, cmap=plt.cm.Paired);  #
Adjusted cmap and variable names

# Add labels and title
plt.xlabel('Feature_A')
plt.ylabel('Feature_B')
plt.title('Binary Classification Dataset')

plt.show()
```

- import matplotlib.pyplot as plt: Mengimpor library matplotlib dengan alias plt.
- import numpy as np: Mengimpor library numpy dengan alias np.
- plt.scatter(X_new[:, 0], X_new[:, 1], c=y_new, cmap=plt.cm.Paired): Membuat scatter plot dengan menggunakan features dari dataset baru (X_new). Sumbu x dan y masing-masing diambil dari kolom pertama ([:, 0]) dan kolom kedua ([:, 1]) dari features. Warna titik pada plot ditentukan oleh labels (y_new), dengan menggunakan colormap plt.cm.Paired.
- plt.xlabel('Feature_A') dan plt.ylabel('Feature_B'): Menambahkan label sumbu x dan y masing-masing dengan teks "Feature_A" dan "Feature_B".
- plt.title('Binary Classification Dataset'): Menambahkan judul plot dengan teks "Binary Classification Dataset".
- plt.show(): Menampilkan plot.

Kode ini bertujuan untuk memvisualisasikan dataset yang baru dibuat dalam bentuk scatter plot menggunakan library matplotlib, untuk memberikan visualisasi scatter plot dari dataset baru, di mana sumbu x dan y mewakili dua fitur dari dataset, dan warna titik pada plot mencerminkan label kelas (0 atau 1) dari dataset. Scatter plot ini memberikan gambaran visual tentang bagaimana data tersebar dan pola kelasnya.

```
import torch
from sklearn.model_selection import train_test_split  # Import
train_test_split
```

```
# Convert data to tensors with different variable names
X_tensor_custom = torch.tensor(X_new, dtype=torch.float)
y_tensor_custom = torch.tensor(y_new, dtype=torch.float)

# Split the data into train and test sets with different variable names
X_train_custom, X_test_custom, y_train_custom, y_test_custom =
train_test_split(X_tensor_custom,

      y_tensor_custom,

      test_size=0.2,

      random_state=RANDOM_SEED_NEW)

# Display the lengths of the resulting sets
print(f"Length of X_train_custom: {len(X_train_custom)}")
print(f"Length of X_test_custom: {len(X_test_custom)}")
print(f"Length of y_train_custom: {len(y_train_custom)}")
print(f"Length of y_test_custom: {len(y_test_custom)}")
```

- import torch: Mengimpor library PyTorch.
- from sklearn.model_selection import train_test_split: Mengimpor fungsi train_test_split dari scikit-learn, yang akan digunakan untuk membagi dataset menjadi set pelatihan dan set pengujian.
- X_tensor_custom = torch.tensor(X_new, dtype=torch.float): Mengonversi features dari dataset baru (X_new) menjadi PyTorch tensor dengan tipe data float32, dan disimpan dalam variabel X_tensor_custom.
- y_tensor_custom = torch.tensor(y_new, dtype=torch.float): Mengonversi labels dari dataset baru (y_new) menjadi PyTorch tensor dengan tipe data float32, dan disimpan dalam variabel y_tensor_custom.
- X_train_custom, X_test_custom, y_train_custom, y_test_custom = train_test_split(...):
- train_test_split(X_tensor_custom, y_tensor_custom, test_size=0.2, random_state=RANDOM_SEED_NEW): Membagi dataset menjadi set pelatihan dan set pengujian dengan proporsi 80-20% dan menggunakan seed pengacakan (random_state) yang baru (RANDOM_SEED_NEW). Hasilnya adalah empat variabel: X_train_custom (features set pelatihan), X_test_custom (features set pengujian), y_train_custom (labels set pelatihan), dan y_test_custom (labels set pengujian).

- print(f"Length of X_train_custom: {len(X_train_custom)}") dan sejenisnya: Menampilkan panjang (jumlah sampel) dari set pelatihan dan set pengujian.

Kode ini adalah langkah-langkah untuk mengonversi data dari dataset baru menjadi PyTorch tensors, dan kemudian membagi dataset menjadi set pelatihan (train set) dan set pengujian (test set) menggunakan fungsi train_test_split dari scikit-learn, untuk mengonversi dataset baru menjadi PyTorch tensors dan membaginya menjadi set pelatihan dan set pengujian menggunakan train_test_split. Hal ini berguna untuk melatih dan menguji model machine learning pada dataset yang berbeda secara terpisah.

```
import torch
from torch import nn

class CustomModelV1(nn.Module):
    def __init__(self, input_size, output_size, hidden_size):
        super().__init__()

        self.fc_first = nn.Linear(in_features=input_size,
                                  out_features=hidden_size)
        self.fc_second = nn.Linear(in_features=hidden_size,
                                   out_features=hidden_size)
        self.fc_output = nn.Linear(in_features=hidden_size,
                                   out_features=output_size)
        self.activation = nn.ReLU()

    def forward(self, x):
        return
self.fc_output(self.activation(self.fc_second(self.activation(self.fc_firs
t(x)))))

# Define the input, output, and hidden sizes
input_size_custom = 2
output_size_custom = 1
hidden_size_custom = 12  # Different hidden layer size

# Instantiate the custom model
model_custom_v1 = CustomModelV1(input_size_custom, output_size_custom,
hidden_size_custom).to(device)
model_custom_v1
```

- import torch dan from torch import nn: Mengimpor library PyTorch dan modul nn (neural networks).
- class CustomModelV1(nn.Module):: Membuat kelas CustomModelV1 yang merupakan turunan dari kelas nn.Module. Ini adalah kelas dasar untuk semua model PyTorch.
- def __init__(self, input_size, output_size, hidden_size):: Metode konstruktor (__init__) untuk menginisialisasi objek model. Menerima tiga parameter: input_size (ukuran input), output_size (ukuran output), dan hidden_size (ukuran lapisan tersembunyi).
- super().__init__(): Memanggil konstruktor dari kelas induk (nn.Module) untuk menginisialisasi bagian dasar dari model.
- self.fc_first = nn.Linear(in_features=input_size, out_features=hidden_size): Membuat lapisan linear pertama (fully connected) dengan jumlah input sesuai dengan input_size dan jumlah output sesuai dengan hidden_size.
- self.fc_second = nn.Linear(in_features=hidden_size, out_features=hidden_size): Membuat lapisan linear kedua dengan jumlah input dan output sama, yaitu hidden_size.
- self.fc_output = nn.Linear(in_features=hidden_size, out_features=output_size): Membuat lapisan output dengan jumlah input sesuai dengan hidden_size dan jumlah output sesuai dengan output_size.
- self.activation = nn.ReLU(): Membuat fungsi aktivasi ReLU untuk digunakan di antara lapisan-lapisan linear.
- def forward(self, x):: Metode forward yang mendefinisikan bagaimana data mengalir melalui model. Dalam hal ini, data x melewati lapisan-lapisan linear dan fungsi aktivasi secara berurutan.
- return self.fc_output(self.activation(self.fc_second(self.activation(self.fc_first(x))))): Menggunakan lapisan linear, fungsi aktivasi ReLU, dan urutan yang telah ditentukan untuk menghitung output model.
- input_size_custom = 2, output_size_custom = 1, hidden_size_custom = 12: Menetapkan ukuran input, output, dan lapisan tersembunyi untuk model.
- model_custom_v1 = CustomModelV1(input_size_custom, output_size_custom, hidden_size_custom).to(device): Menginstansiasi model CustomModelV1 dengan ukuran input, output, dan lapisan tersembunyi yang telah ditetapkan, dan memindahkan model ke perangkat yang ditentukan (CPU atau GPU) menggunakan metode .to(device).

- model_custom_v1: Menampilkan struktur model yang telah dibuat.

Kode ini mendefinisikan sebuah model neural network sederhana menggunakan PyTorch dengan satu lapisan tersembunyi (hidden layer), untuk mendefinisikan dan menginstansiasi model neural network sederhana dengan satu lapisan tersembunyi menggunakan PyTorch. Model ini dapat digunakan untuk tugas klasifikasi biner dengan ukuran input 2 dan ukuran output 1.

```
# Create an instance of the custom model
model_custom_v1 = CustomModelV1(input_size_custom, output_size_custom,
hidden_size_custom).to(device)

# Display the state dictionary of the custom model
model_custom_v1.state_dict()
```

- model_custom_v1 = CustomModelV1(input_size_custom, output_size_custom, hidden_size_custom).to(device): Menginstansiasi kembali model CustomModelV1 dengan menggunakan ukuran input, output, dan lapisan tersembunyi yang telah ditetapkan sebelumnya. Model ini kemudian dipindahkan ke perangkat yang telah ditentukan (CPU atau GPU) menggunakan metode .to(device).
- model_custom_v1.state_dict(): Memanggil metode state_dict() pada model untuk mengambil dictionary yang berisi parameter-parameter model. State dictionary ini berisi informasi tentang parameter (bobot dan bias) dari setiap lapisan dalam model.

Kode tersebut membuat sebuah instance dari model yang telah didefinisikan sebelumnya (CustomModelV1) dan menampilkan dictionary (state dictionary) yang berisi parameter-parameter model, untuk menampilkan state dictionary dari model. State dictionary ini berguna jika kita ingin menyimpan atau memuat kembali parameter-parameter model, atau untuk melihat nilai parameter saat ini.

```
# Choose the Binary Cross Entropy with Logits Loss
custom_loss_fn = nn.BCEWithLogitsLoss()

# Choose the Stochastic Gradient Descent optimizer
custom_optimizer = torch.optim.SGD(params=model_custom_v1.parameters(),
                                   lr=0.05)  # Adjusted learning rate
```

- custom_loss_fn = nn.BCEWithLogitsLoss(): Memilih fungsi kerugian Binary Cross Entropy with Logits Loss. Fungsi ini umumnya digunakan untuk tugas klasifikasi biner pada model yang memberikan output dalam bentuk logits (tanpa fungsi aktivasi seperti sigmoid di akhir).
- custom_optimizer = torch.optim.SGD(params=model_custom_v1.parameters(), lr=0.05): Memilih optimizer Stochastic Gradient Descent (SGD) dengan laju pembelajaran (learning rate) sebesar 0.05. Optimizer ini akan mengoptimalkan parameter-parameter model (model_custom_v1.parameters()) menggunakan algoritma Stochastic Gradient Descent.

Kode ini menentukan fungsi kerugian (loss function) dan optimizer untuk model yang telah dibuat, untuk memilih dan mengkonfigurasi fungsi kerugian dan optimizer yang akan digunakan selama pelatihan model. Pilihan ini mempengaruhi bagaimana model diperbarui selama proses pelatihan.

```
# Logits (raw outputs of the custom model)
custom_logits = model_custom_v1(X_train_custom.to(device)[:10]).squeeze()
print("Custom Logits:")
print(custom_logits)

# Prediction probabilities
custom_pred_probs = torch.sigmoid(custom_logits)
print("Custom Pred Probs:")
print(custom_pred_probs)

# Predicted labels (rounded prediction probabilities)
custom_pred_labels = torch.round(custom_pred_probs)
print("Custom Pred Labels:")
print(custom_pred_labels)
```

- custom_logits = model_custom_v1(X_train_custom.to(device)[:10]).squeeze(): Menghitung logits atau output raw dari model untuk sepuluh sampel pertama dari set pelatihan (X_train_custom). Metode .squeeze() digunakan untuk menghilangkan dimensi yang memiliki ukuran 1, sehingga hasilnya menjadi tensor 1D.
- print("Custom Logits:") dan print(custom_logits): Menampilkan logits yang dihasilkan oleh model untuk sepuluh sampel pertama.
- custom_pred_probs = torch.sigmoid(custom_logits): Menggunakan fungsi sigmoid untuk mengonversi logits menjadi probabilitas. Probabilitas ini mengindikasikan sejauh mana model yakin bahwa setiap sampel milik kelas positif (1).
- print("Custom Pred Probs:") dan print(custom_pred_probs): Menampilkan probabilitas prediksi untuk sepuluh sampel pertama.
- custom_pred_labels = torch.round(custom_pred_probs): Menghasilkan label prediksi dengan membulatkan probabilitas prediksi. Jika probabilitas lebih besar dari 0.5, dianggap sebagai kelas positif (1), dan sebaliknya.
- print("Custom Pred Labels:") dan print(custom_pred_labels): Menampilkan label prediksi yang dihasilkan.

Kode tersebut melakukan prediksi menggunakan model yang telah dibuat untuk menghasilkan dan mengevaluasi prediksi dari model untuk sepuluh sampel pertama dari set pelatihan. Hal ini membantu dalam memahami bagaimana model memberikan output logit, probabilitas, dan label prediksi untuk sampel-sampel tertentu.

```
# 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
```

- !pip -q install torchmetrics: Menginstal library torchmetrics jika belum terinstal. Perhatikan bahwa !pip adalah cara untuk menjalankan perintah pip dari dalam notebook atau lingkungan yang mendukung shell.
- from torchmetrics import Accuracy: Mengimpor metrik akurasi (Accuracy) dari library torchmetrics.
- acc_fn = Accuracy(task="multiclass", num_classes=2).to(device): Membuat instance dari metrik akurasi dengan menyertakan parameter task yang diatur ke "multiclass" dan num_classes diatur ke 2. Metrik ini kemudian dipindahkan ke perangkat yang telah ditentukan sebelumnya (device).
- acc_fn: Menampilkan instance metrik akurasi yang telah dibuat.

Kode tersebut memiliki dua tujuan utama: menginstal library torchmetrics (jika belum terinstal) dan membuat instance dari metrik akurasi, untuk menginstal library torchmetrics dan membuat instance dari metrik akurasi yang akan digunakan untuk mengevaluasi kinerja model. Metrik ini kemudian dapat digunakan untuk menghitung akurasi model berdasarkan prediksi dan label sebenarnya.

```
import torch

# Set random seed for reproducibility
torch.manual_seed(123)

# Number of epochs
num_epochs = 1000

# Send data to the device
X_train_custom, y_train_custom = X_train_custom.to(device),
y_train_custom.to(device)
X_test, y_test = X_test_custom.to(device), y_test_custom.to(device)

# Loop through the data
for epoch in range(num_epochs):
    ### Training
    model_custom_v1.train()

    # 1. Forward pass
    train_logits = model_custom_v1(X_train_custom).squeeze()
    train_pred_probs = torch.sigmoid(train_logits)
    train_pred = torch.round(train_pred_probs)

    # 2. Calculate the loss
```

```
    train_loss = custom_loss_fn(train_logits, y_train_custom)
    train_acc = acc_fn(train_pred, y_train_custom.int())

    # 3. Zero the gradients
    custom_optimizer.zero_grad()

    # 4. Backward pass (backpropagation)
    train_loss.backward()

    # 5. Update weights
    custom_optimizer.step()

    ### Testing
    model_custom_v1.eval()
    with torch.no_grad():
        # 1. Forward pass
        test_logits = model_custom_v1(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))

        # 2. Calculate the loss/accuracy
        test_loss = custom_loss_fn(test_logits, y_test)
        test_acc = acc_fn(test_pred, y_test.int())

    # Print out the progress
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Train Loss: {train_loss:.2f} Train Acc:
{train_acc:.2f} | Test Loss: {test_loss:.2f} Test Acc: {test_acc:.2f}")
```

- torch.manual_seed(123): Menetapkan seed untuk generator bilangan acak PyTorch, sehingga hasil eksperimen dapat direproduksi.
- num_epochs = 1000: Menetapkan jumlah epoch (iterasi melalui seluruh dataset pelatihan) yang akan dilakukan selama pelatihan model.
- X_train_custom, y_train_custom = X_train_custom.to(device), y_train_custom.to(device): Memindahkan data pelatihan ke perangkat yang telah ditentukan.
- X_test, y_test = X_test_custom.to(device), y_test_custom.to(device): Memindahkan data pengujian ke perangkat yang telah ditentukan.
- Looping melalui epoch:
  - model_custom_v1.train(): Mengatur model ke mode pelatihan.

- Forward pass pada data pelatihan untuk menghitung logits, probabilitas prediksi, dan prediksi.
- Menghitung loss dan akurasi pada data pelatihan.
- Mengatur gradien model menjadi nol.
- Melakukan backward pass untuk menghitung gradien loss terhadap parameter model.
- Mengupdate parameter model menggunakan optimizer.
- model_custom_v1.eval(): Mengatur model ke mode evaluasi.
- Forward pass pada data pengujian untuk menghitung prediksi.
- Menghitung loss dan akurasi pada data pengujian.
- Mencetak hasil setiap 100 epoch untuk melihat perkembangan pelatihan.

Kode ini adalah implementasi pelatihan (training) dan evaluasi (testing) model menggunakan loop for sepanjang beberapa epoch, melakukan pelatihan dan pengujian model dengan mengulang proses forward pass, backward pass, dan pembaruan parameter model sepanjang beberapa epoch. Hal ini memberikan informasi tentang seberapa baik model dapat mempelajari pola dari data pelatihan dan seberapa baik dapat menggeneralisasi ke data pengujian.

```
import numpy as np

# TK - this could go in the helper_functions.py and be explained there
def plot_decision_boundary_v2(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() - 0.2, X[:, 0].max() + 0.2
    y_min, y_max = X[:, 1].min() - 0.2, X[:, 1].max() + 0.2
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 151),
                         np.linspace(y_min, y_max, 151))

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

    # Make predictions
```

```
    model.eval()
    with torch.no_grad():
        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.GnBu, alpha=0.7)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.GnBu)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("Decision Boundary")

# Usage example:
# plot_decision_boundary_v2(model_0, X_test, y_test)
# plt.show()
```

- def plot_decision_boundary_v2(model, X, y):: Mendefinisikan fungsi dengan tiga parameter: model (model yang akan digunakan), X (features), dan y (labels).
- model.to("cpu"), X, y = X.to("cpu"), y.to("cpu"): Memastikan bahwa semua data (model, features, dan labels) berada di CPU. Ini dilakukan untuk konsistensi dengan NumPy dan Matplotlib.
- x_min, x_max = X[:, 0].min() - 0.2, X[:, 0].max() + 0.2: Menentukan nilai minimum dan maksimum untuk sumbu x pada plot batas keputusan dengan menambahkan jarak 0.2 dari setiap sisi.
- xx, yy = np.meshgrid(np.linspace(x_min, x_max, 151), np.linspace(y_min, y_max, 151)): Membuat grid dari nilai-nilai x dan y yang dihasilkan oleh fungsi linspace dari NumPy.

- X_to_pred_on = torch.from_numpy(np.column_stack((xx.ravel(), yy.ravel()))).float(): Membuat features untuk diprediksi oleh model menggunakan grid nilai x dan y yang telah dibuat sebelumnya.
- model.eval(), with torch.no_grad(): y_logits = model(X_to_pred_on): Menyatakan model dalam mode evaluasi dan menghitung logits (output sebelum fungsi aktivasi) untuk setiap titik pada grid menggunakan model yang telah dilatih.
- Menentukan prediksi berdasarkan apakah masalah klasifikasi multikelas atau biner:
  - Jika jumlah kelas lebih dari 2 (len(torch.unique(y)) > 2), maka menggunakan argmax pada softmax dari logits.
  - Jika jumlah kelas sama dengan 2, maka menggunakan fungsi sigmoid untuk menghasilkan prediksi biner.
- Menggambarkan batas keputusan dengan menggunakan kontur plot (plt.contourf) dengan warna yang sesuai dengan hasil prediksi. Scatter plot juga ditambahkan untuk menunjukkan distribusi data asli.
- Memberikan label sumbu x, sumbu y, dan judul plot.
- plt.show(): Menampilkan plot.

Kode tersebut mendefinisikan fungsi plot_decision_boundary_v2 yang digunakan untuk menggambar batas keputusan (decision boundary) dari model pada data dengan dua fitur. Fungsi ini kemudian memberikan contoh penggunaannya. Fungsi ini berguna untuk visualisasi batas keputusan dari model klasifikasi biner atau multikelas pada data dua dimensi. Contoh penggunaannya diberikan sebagai komentar pada akhir fungsi.

```
# Plot decision boundaries for training and test sets
plt.figure(figsize=(14, 7))
plt.subplot(1, 2, 1)
plt.title("Decision Boundaries - Training Data")
plot_decision_boundary_v2(model_custom_v1, X_train_custom, y_train_custom)

plt.subplot(1, 2, 2)
plt.title("Decision Boundaries - Test Data")
plot_decision_boundary_v2(model_custom_v1, X_test, y_test)

plt.show()
```

- plt.figure(figsize=(14, 7)): Membuat objek gambar matplotlib dengan ukuran figur (figure size) 14x7.
- plt.subplot(1, 2, 1): Membuat subplot pertama (dari dua subplot) dengan 1 baris dan 2 kolom, dan memilih subplot pertama.
- plt.title("Decision Boundaries - Training Data"): Memberikan judul untuk subplot pertama.
- plot_decision_boundary_v2(model_custom_v1, X_train_custom, y_train_custom): Memanggil fungsi plot_decision_boundary_v2 untuk menggambar batas keputusan pada data pelatihan menggunakan model model_custom_v1.
- plt.subplot(1, 2, 2): Membuat subplot kedua dan memilihnya.
- plt.title("Decision Boundaries - Test Data"): Memberikan judul untuk subplot kedua.
- plot_decision_boundary_v2(model_custom_v1, X_test, y_test): Memanggil fungsi plot_decision_boundary_v2 untuk menggambar batas keputusan pada data pengujian menggunakan model yang sama.
- plt.show(): Menampilkan plot dengan dua subplot yang menggambarkan batas keputusan pada data pelatihan dan pengujian.

Kode ini menggunakan fungsi plot_decision_boundary_v2 untuk menggambarkan batas keputusan (decision boundaries) dari model pada data pelatihan dan data pengujian, untuk membuat subplot dengan dua gambar yang menunjukkan batas keputusan dari model pada data pelatihan dan pengujian. Hal ini membantu dalam memvisualisasikan bagaimana model mengklasifikasikan area pada ruang fitur.

```
# Generate a tensor and plot it
tensor_A = torch.arange(-50, 50, 0.5)
plt.plot(tensor_B)
plt.xlabel('Index')
plt.ylabel('Values')
plt.title('Plot of Tensor A')
plt.show()
```

- tensor_A = torch.arange(-50, 50, 0.5): Membuat tensor tensor_A dengan menggunakan fungsi torch.arange(). Tensor ini akan berisi nilai dari -50 hingga 50 dengan jarak 0.5 antara setiap nilai.
- plt.plot(tensor_A): Memplot nilai-nilai dari tensor tensor_A. Nilai-nilai ini akan diplot terhadap indeks tensor, dan garis yang menghubungkan nilai-nilai tersebut akan dibuat.
- plt.xlabel('Index'): Menambahkan label sumbu x dengan teks "Index".
- plt.ylabel('Values'): Menambahkan label sumbu y dengan teks "Values".
- plt.title('Plot of Tensor A'): Menambahkan judul plot dengan teks "Plot of Tensor A".
- plt.show(): Menampilkan plot.

Kode ini menciptakan dan memplot tensor yang disebut tensor_A untuk membuat tensor dan memplot nilainya terhadap indeks tensor. Hal ini berguna untuk memvisualisasikan pola atau distribusi nilai dalam tensor. Namun, ada ketidakcocokan antara nama tensor (tensor_A) dan nama yang digunakan dalam fungsi plot (tensor_B). Jadi, seharusnya fungsi plot menggunakan nama yang benar, yaitu tensor_A.

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

- torch.tanh(tensor_A): Menghitung tangen hiperbolik dari setiap elemen dalam tensor tensor_A.
- plt.plot(...): Memplot hasil dari tangen hiperbolik tensor tensor_A. Grafik ini akan menunjukkan bagaimana nilai-nilai dalam tensor diubah oleh fungsi tangen hiperbolik.
- plt.show(): Menampilkan plot.

Kode ini memplot fungsi tangen hiperbolik (tanh) dari tensor tensor_A untuk memvisualisasikan efek dari fungsi tangen hiperbolik pada nilai-nilai tensor. Fungsi tangen hiperbolik menghasilkan output yang terletak dalam rentang (-1, 1), dan plot ini memberikan gambaran tentang transformasi non-linear ini terhadap nilai-nilai tensor awal.

```
# Define the tanh function and plot it
def custom_tanh(x):
```

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

plt.plot(custom_tanh(tensor_A))

plt.show()
```

- def custom_tanh(x):: Mendefinisikan fungsi tangen hiperbolik kustom dengan parameter x.
- return (torch.exp(x) - torch.exp(-x)) / (torch.exp(x) + torch.exp(-x)): Implementasi rumus tangen hiperbolik menggunakan fungsi eksponensial PyTorch (torch.exp). Fungsi ini mengembalikan nilai tangen hiperbolik dari input x.
- plt.plot(custom_tanh(tensor_A)): Memplot hasil dari fungsi tangen hiperbolik kustom (custom_tanh) dengan menggunakan tensor tensor_A. Grafik ini akan menunjukkan bagaimana nilai-nilai dalam tensor diubah oleh fungsi tangen hiperbolik kustom.
- plt.show(): Menampilkan plot.

Kode tersebut mendefinisikan fungsi tangen hiperbolik kustom (custom_tanh) dan memplot hasilnya menggunakan tensor tensor_A untuk memvisualisasikan hasil dari fungsi tangen hiperbolik kustom yang telah didefinisikan. Fungsi tangen hiperbolik kustom ini mengimplementasikan rumus matematika tangen hiperbolik secara manual menggunakan fungsi eksponensial PyTorch. Plot ini memberikan gambaran tentang transformasi non-linear terhadap nilai-nilai tensor menggunakan fungsi tangen hiperbolik kustom.

```
import numpy as np
import matplotlib.pyplot as plt

CUSTOM_RANDOM_SEED = 42
np.random.seed(CUSTOM_RANDOM_SEED)

CUSTOM_N = 100  # number of points per class
CUSTOM_D = 2    # dimensionality
CUSTOM_K = 3    # number of classes

custom_X = np.zeros((CUSTOM_N * CUSTOM_K, CUSTOM_D))  # data matrix (each
row = single example)
```

```
custom_y = np.zeros(CUSTOM_N * CUSTOM_K, dtype='uint8')  # class labels

for j in range(CUSTOM_K):
    custom_ix = range(CUSTOM_N * j, CUSTOM_N * (j + 1))
    custom_r = np.linspace(0.0, 1, CUSTOM_N)  # radius
    custom_t = np.linspace(j * 4, (j + 1) * 4, CUSTOM_N) +
np.random.randn(CUSTOM_N) * 0.2  # theta
    custom_X[custom_ix] = np.c_[custom_r * np.sin(custom_t), custom_r *
np.cos(custom_t)]
    custom_y[custom_ix] = j

# Visualize the data
plt.scatter(custom_X[:, 0], custom_X[:, 1], c=custom_y, s=40,
cmap=plt.cm.RdYlBu)
plt.title('Custom Spiral Dataset (Different Seed and Values)')
plt.show()
```

- CUSTOM_RANDOM_SEED = 42: Menetapkan seed untuk generator bilangan acak NumPy untuk memastikan hasil yang dapat direproduksi.
- np.random.seed(CUSTOM_RANDOM_SEED): Menetapkan seed generator bilangan acak NumPy.
- CUSTOM_N = 100, CUSTOM_D = 2, CUSTOM_K = 3: Menetapkan beberapa parameter untuk dataset, seperti jumlah titik per kelas (CUSTOM_N), dimensi data (CUSTOM_D), dan jumlah kelas (CUSTOM_K).
- custom_X = np.zeros((CUSTOM_N * CUSTOM_K, CUSTOM_D)): Membuat matriks kosong custom_X dengan ukuran (jumlah titik total, dimensi data).
- custom_y = np.zeros(CUSTOM_N * CUSTOM_K, dtype='uint8'): Membuat array kosong custom_y untuk menyimpan label kelas.
- Loop for j in range(CUSTOM_K): Membuat data untuk setiap kelas spiral:
  - custom_ix = range(CUSTOM_N * j, CUSTOM_N * (j + 1)): Menentukan indeks untuk setiap kelas.
  - custom_r = np.linspace(0.0, 1, CUSTOM_N): Membuat urutan nilai radius dari 0 hingga 1.
  - custom_t = np.linspace(j * 4, (j + 1) * 4, CUSTOM_N) + np.random.randn(CUSTOM_N) * 0.2: Membuat urutan nilai theta (sudut) yang membentuk spiral dengan noise yang ditambahkan.

    - custom_X[custom_ix] = np.c_[custom_r * np.sin(custom_t), custom_r * np.cos(custom_t)]: Mengisi matriks custom_X dengan nilai-nilai koordinat (x, y) yang membentuk spiral.
    - custom_y[custom_ix] = j: Mengisi array custom_y dengan label kelas.
- plt.scatter(custom_X[:, 0], custom_X[:, 1], c=custom_y, s=40, cmap=plt.cm.RdYlBu): Memvisualisasikan dataset dengan menggunakan scatter plot, di mana setiap kelas ditandai dengan warna yang berbeda. Parameter s menentukan ukuran marker.
- plt.title('Custom Spiral Dataset (Different Seed and Values)'): Menambahkan judul plot.
- plt.show(): Menampilkan plot.

Kode ini membuat dataset sintetis yang disebut "Custom Spiral Dataset" yang terdiri dari tiga kelas spiral yang berbeda untuk membuat dan memvisualisasikan dataset sintetis dengan tiga kelas spiral yang berbeda. Setiap kelas ditandai dengan warna yang berbeda pada scatter plot. Dataset ini berguna untuk memahami bagaimana model dapat memahami pola dan membedakan kelas-kelas yang berbeda.

```
# Turn custom data into tensors
custom_X_tensor_v2 = torch.from_numpy(custom_X).type(torch.float)  #
features as float32
custom_y_tensor_v2 = torch.from_numpy(custom_y).type(torch.LongTensor)  #
labels need to be of type long

# Create custom train and test splits
from sklearn.model_selection import train_test_split
custom_X_train_v2, custom_X_test_v2, custom_y_train_v2, custom_y_test_v2 =
train_test_split(custom_X_tensor_v2,

                 custom_y_tensor_v2,

                 test_size=0.2,

                 random_state=CUSTOM_RANDOM_SEED)
len(custom_X_train_v2), len(custom_X_test_v2), len(custom_y_train_v2),
len(custom_y_test_v2)
```

- custom_X_tensor_v2 = torch.from_numpy(custom_X).type(torch.float): Mengkonversi array NumPy custom_X ke tensor PyTorch dan mengubah tipe datanya menjadi torch.float (float32).
- custom_y_tensor_v2 = torch.from_numpy(custom_y).type(torch.LongTensor): Mengkonversi array NumPy custom_y ke tensor PyTorch dan mengubah tipe datanya menjadi torch.LongTensor. Label kelas perlu memiliki tipe data long untuk melibatkan fungsi loss dalam pelatihan model.
- from sklearn.model_selection import train_test_split: Mengimpor fungsi train_test_split dari library scikit-learn untuk membuat pembagian data antara data pelatihan dan pengujian.
- custom_X_train_v2, custom_X_test_v2, custom_y_train_v2, custom_y_test_v2 = train_test_split(...):
  - Melakukan pembagian data menggunakan train_test_split dengan parameter tensor custom_X_tensor_v2 dan custom_y_tensor_v2.
  - test_size=0.2 menentukan bahwa 20% dari data akan digunakan untuk pengujian.
  - random_state=CUSTOM_RANDOM_SEED memastikan bahwa pembagian data bersifat deterministik dengan seed yang telah ditetapkan sebelumnya.
- len(custom_X_train_v2), len(custom_X_test_v2), len(custom_y_train_v2), len(custom_y_test_v2): Menampilkan panjang (jumlah sampel) dari setiap set data, yaitu data pelatihan dan data pengujian.

Kode ini melakukan konversi dataset spiral kustom ke dalam bentuk tensor PyTorch dan membuat pembagian data antara data pelatihan dan data pengujian, untuk mengonversi dataset kustom ke dalam bentuk tensor PyTorch dan membuat pembagian data antara data pelatihan dan pengujian. Hal ini diperlukan untuk menggunakan dataset ini dalam pelatihan dan evaluasi model PyTorch.

```
# Create an instance of the Accuracy metric for multi-class classification
accuracy_metric_multi = Accuracy(task="multiclass",
num_classes=3).to(device)
accuracy_metric_multi
```

- accuracy_metric_multi = Accuracy(task="multiclass", num_classes=3).to(device): Membuat instance dari metrik akurasi dengan menyertakan parameter task yang diatur ke "multiclass" dan num_classes diatur ke 3. Metrik ini kemudian dipindahkan ke perangkat yang telah ditentukan sebelumnya (device).
  - task="multiclass": Menentukan bahwa metrik ini digunakan untuk tugas klasifikasi multikelas.
  - num_classes=3: Menentukan jumlah kelas yang ada dalam tugas klasifikasi multikelas ini.
- accuracy_metric_multi: Menampilkan instance metrik akurasi yang telah dibuat.

Kode tersebut membuat instance dari metrik akurasi (Accuracy) untuk tugas klasifikasi multikelas. Metrik akurasi digunakan untuk mengukur sejauh mana model dapat mengklasifikasikan dengan benar sampel-sampel dalam dataset, dan penggunaannya dapat berbeda tergantung pada jenis tugas klasifikasi yang sedang dilakukan (biner atau multikelas). Metrik akurasi ini akan berguna untuk mengevaluasi kinerja model pada tugas klasifikasi multikelas dengan 3 kelas.

```
# Check for GPU using PyTorch
device_spiral = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print(f"Using device: {device_spiral}")

class CustomSpiralModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(in_features=2, out_features=10)
        self.layer2 = nn.Linear(in_features=10, out_features=10)
        self.layer3 = nn.Linear(in_features=10, out_features=3)
        self.relu = nn.ReLU()

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

# Instantiate the model and move it to the target device
model_custom_spiral = CustomSpiralModel().to(device_spiral)
model_custom_spiral
```

- device_spiral = torch.device("cuda" if torch.cuda.is_available() else "cpu"): Memeriksa apakah GPU (cuda) tersedia. Jika ya, maka device_spiral diatur ke "cuda", jika tidak, diatur ke "cpu". Variabel device_spiral nantinya akan digunakan untuk memindahkan model dan data ke perangkat yang sesuai.
- print(f"Using device: {device_spiral}"): Mencetak perangkat yang akan digunakan (cuda atau cpu) untuk pelatihan model.
- class CustomSpiralModel(nn.Module):: Mendefinisikan kelas CustomSpiralModel yang merupakan subclass dari nn.Module. Kelas ini akan menjadi blueprint untuk arsitektur model jaringan saraf.
  - __init__: Menginisialisasi layer-layer jaringan saraf, terdiri dari tiga layer linear dan fungsi aktivasi ReLU.
  - forward: Mendefinisikan aliran maju (forward pass) dari model, yaitu bagaimana input propagasi melalui layer-layer untuk menghasilkan output.
- model_custom_spiral = CustomSpiralModel().to(device_spiral): Menginisialisasi instance dari model CustomSpiralModel dan memindahkan model ke perangkat yang telah ditentukan (device_spiral).
- model_custom_spiral: Menampilkan arsitektur model yang telah dibuat.

Kode ini memeriksa ketersediaan GPU menggunakan PyTorch dan mendefinisikan serta menginisialisasi model arsitektur jaringan saraf untuk tugas klasifikasi multikelas pada dataset spiral untuk memeriksa ketersediaan GPU, mendefinisikan arsitektur model jaringan saraf untuk tugas klasifikasi multikelas pada dataset spiral, dan memindahkan model ke perangkat yang sesuai. Model ini terdiri dari tiga layer linear dan fungsi aktivasi ReLU untuk menangani data yang berasal dari dataset spiral dengan dua fitur.

```
# Setup data to be device agnostic
X_train_custom, y_train_custom = X_train_custom.to(device_spiral),
y_train_custom.to(device_spiral)
X_test_custom, y_test_custom = X_test.to(device_spiral),
y_test.to(device_spiral)
print(X_train_custom.dtype, X_test_custom.dtype, y_train_custom.dtype,
y_test_custom.dtype)

# Print out untrained custom model outputs
print("Custom Logits:")
print(model_custom_spiral(X_train_custom)[:10])
```

```
print("Custom Pred probs:")
print(torch.softmax(model_custom_spiral(X_train_custom)[:10], dim=1))

print("Custom Pred labels:")
print(torch.softmax(model_custom_spiral(X_train_custom)[:10],
dim=1).argmax(dim=1))
```

- X_train_custom, y_train_custom = X_train_custom.to(device_spiral), y_train_custom.to(device_spiral): Memindahkan tensor-tensor data pelatihan (X_train_custom dan y_train_custom) ke perangkat yang telah ditentukan (device_spiral).
- X_test_custom, y_test_custom = X_test.to(device_spiral), y_test.to(device_spiral): Memindahkan tensor-tensor data pengujian (X_test_custom dan y_test_custom) ke perangkat yang telah ditentukan (device_spiral).
- print(X_train_custom.dtype, X_test_custom.dtype, y_train_custom.dtype, y_test_custom.dtype): Mencetak tipe data (dtype) dari tensor-tensor yang telah dipindahkan ke perangkat. Ini memberikan informasi tentang apakah data telah berhasil dipindahkan dan apakah tipe datanya sesuai dengan yang diharapkan.
- print("Custom Logits:"): Mencetak output logit (sebelum fungsi aktivasi) dari model untuk 10 sampel pertama dari data pelatihan.
- print(model_custom_spiral(X_train_custom)[:10]): Mencetak nilai logit dari model untuk 10 sampel pertama dari data pelatihan.
- print("Custom Pred probs:"): Mencetak probabilitas hasil prediksi dari model untuk 10 sampel pertama dari data pelatihan.
- print(torch.softmax(model_custom_spiral(X_train_custom)[:10], dim=1)): Mencetak probabilitas hasil prediksi menggunakan fungsi softmax untuk 10 sampel pertama dari data pelatihan.
- print("Custom Pred labels:"): Mencetak label hasil prediksi dari model untuk 10 sampel pertama dari data pelatihan.
- print(torch.softmax(model_custom_spiral(X_train_custom)[:10], dim=1).argmax(dim=1)): Mencetak label hasil prediksi menggunakan argmax pada hasil softmax untuk 10 sampel pertama dari data pelatihan.

Kode ini mengatur data untuk dapat berjalan di perangkat mana pun (device agnostic), memindahkan data ke perangkat yang sesuai, dan mencetak output model yang belum dilatih untuk memastikan bahwa data telah dipindahkan ke perangkat yang benar dan mencetak output model yang belum dilatih untuk memastikan bahwa model berfungsi dengan baik sebelum proses pelatihan dimulai.

```
# Setup loss function and optimizer for the custom model
custom_loss_fn_spiral = nn.CrossEntropyLoss()
custom_optimizer_spiral =
torch.optim.Adam(model_custom_spiral.parameters(),
                                            lr=0.03)  # Different learning
rate for variety
```

- custom_loss_fn_spiral = nn.CrossEntropyLoss(): Mendefinisikan fungsi kerugian CrossEntropyLoss. Fungsi ini cocok digunakan untuk tugas klasifikasi multikelas, di mana model perlu menghasilkan distribusi probabilitas untuk setiap kelas dan fungsi kerugian akan menghitung kerugian antara distribusi yang dihasilkan dan label yang sebenarnya.
- custom_optimizer_spiral = torch.optim.Adam(model_custom_spiral.parameters(), lr=0.03): Mendefinisikan optimizer Adam untuk memperbarui parameter-model selama pelatihan. Parameter model_custom_spiral.parameters() menyediakan parameter-model yang akan dioptimalkan oleh optimizer. Learning rate (lr) diatur ke 0.03.

Kode ini menyiapkan fungsi kerugian (loss function) dan pengoptimal (optimizer) untuk pelatihan model klasifikasi multikelas pada dataset spiral untuk menentukan fungsi kerugian dan optimizer yang akan digunakan selama pelatihan model pada dataset spiral. CrossEntropyLoss umumnya digunakan untuk tugas klasifikasi multikelas, dan Adam adalah salah satu optimizer yang efektif untuk melatih model neural network.

```
# Loop over data
for custom_epoch_spiral in range(custom_epochs_spiral):
    ## Training
    model_custom_spiral.train()
```

```
    # 1. Forward pass
    custom_y_logits_spiral = model_custom_spiral(X_train_custom)
    custom_y_pred_spiral = torch.softmax(custom_y_logits_spiral,
dim=1).argmax(dim=1)

    # Ensure that both the model output and target have the same datatype
    custom_y_logits_spiral = custom_y_logits_spiral.float()  # Assuming
custom_y_logits_spiral is a tensor of floats
    y_train_custom = y_train_custom.long()  # Assuming y_train_custom is a
tensor of integers

    # 2. Calculate the loss
    loss = custom_loss_fn_spiral(custom_y_logits_spiral, y_train_custom)
    custom_acc_spiral = acc_fn(custom_y_pred_spiral, y_train_custom)

    # 3. Optimizer zero grad
    custom_optimizer_spiral.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    custom_optimizer_spiral.step()

    ## Testing
    model_custom_spiral.eval()
    with torch.no_grad():
        # 1. Forward pass
        custom_test_logits_spiral = model_custom_spiral(X_test_custom)
        custom_test_pred_spiral = torch.softmax(custom_test_logits_spiral,
dim=1).argmax(dim=1)

        # Ensure that both the model output and target have the same
datatype
        custom_test_logits_spiral = custom_test_logits_spiral.float()  #
Assuming custom_test_logits_spiral is a tensor of floats
        y_test_custom = y_test_custom.long()  # Assuming y_test_custom is
a tensor of integers

        # 2. Calculate test loss and acc
        custom_test_loss_spiral =
custom_loss_fn_spiral(custom_test_logits_spiral, y_test_custom)
        custom_test_acc_spiral = acc_fn(custom_test_pred_spiral,
y_test_custom)
```

```
    # Print out what's happening
    if custom_epoch_spiral % 100 == 0:
        print(f"Epoch: {custom_epoch_spiral} | Train loss: {loss:.2f}
Train acc: {custom_acc_spiral:.2f} | "
              f"Test loss: {custom_test_loss_spiral:.2f} Test acc:
{custom_test_acc_spiral:.2f}")
```

- for custom_epoch_spiral in range(custom_epochs_spiral):: Melakukan loop sebanyak custom_epochs_spiral kali untuk iterasi pelatihan.
- model_custom_spiral.train(): Mengatur model dalam mode pelatihan, yang memungkinkan model untuk menghitung gradien dan memperbarui parameter.
- custom_y_logits_spiral = model_custom_spiral(X_train_custom): Melakukan langkah maju (forward pass) untuk mendapatkan logit keluaran dari model untuk data pelatihan.
- custom_y_pred_spiral = torch.softmax(custom_y_logits_spiral, dim=1).argmax(dim=1): Menerapkan fungsi softmax ke logit untuk mendapatkan prediksi kelas. argmax digunakan untuk mendapatkan kelas dengan probabilitas tertinggi.
- Memastikan bahwa baik keluaran model maupun target memiliki tipe data yang sama:
  - custom_y_logits_spiral = custom_y_logits_spiral.float(): Mengubah tipe data logit keluaran model menjadi float.
  - y_train_custom = y_train_custom.long(): Mengubah tipe data target menjadi long (integer).
- loss = custom_loss_fn_spiral(custom_y_logits_spiral, y_train_custom): Menghitung fungsi kerugian menggunakan logit keluaran dan label target.
- custom_acc_spiral = acc_fn(custom_y_pred_spiral, y_train_custom): Menghitung akurasi menggunakan prediksi dan label target.
- custom_optimizer_spiral.zero_grad(): Menetapkan gradien model menjadi nol sebelum langkah mundur (backward pass) pada iterasi berikutnya.
- loss.backward(): Melakukan langkah mundur (backward pass) untuk menghitung gradien parameter.
- custom_optimizer_spiral.step(): Melakukan langkah optimisasi untuk memperbarui parameter-model.

- model_custom_spiral.eval(): Mengatur model dalam mode evaluasi, yang membuat model tidak menghitung atau menyimpan gradien. Ini diperlukan saat melakukan evaluasi pada data pengujian.
- with torch.no_grad():: Menggunakan torch.no_grad() untuk memastikan bahwa operasi di dalam blok ini tidak melibatkan perhitungan gradien.
- custom_test_logits_spiral = model_custom_spiral(X_test_custom): Melakukan langkah maju untuk mendapatkan logit keluaran dari model untuk data pengujian.
- custom_test_pred_spiral = torch.softmax(custom_test_logits_spiral, dim=1).argmax(dim=1): Menerapkan fungsi softmax ke logit untuk mendapatkan prediksi kelas pada data pengujian.
- Memastikan bahwa baik keluaran model pada data pengujian maupun target memiliki tipe data yang sama:
  - custom_test_logits_spiral = custom_test_logits_spiral.float(): Mengubah tipe data logit keluaran model pada data pengujian menjadi float.
  - y_test_custom = y_test_custom.long(): Mengubah tipe data target pada data pengujian menjadi long (integer).
- custom_test_loss_spiral = custom_loss_fn_spiral(custom_test_logits_spiral, y_test_custom): Menghitung fungsi kerugian pada data pengujian.
- custom_test_acc_spiral = acc_fn(custom_test_pred_spiral, y_test_custom): Menghitung akurasi pada data pengujian.
- if custom_epoch_spiral % 100 == 0:: Mencetak hasil pelatihan dan evaluasi setiap 100 iterasi epoch.

Kode ini merupakan loop pelatihan (training loop) dan evaluasi (testing loop) untuk model klasifikasi multikelas pada dataset spiral. Kode ini mencakup langkah-langkah utama dari loop pelatihan dan evaluasi untuk model klasifikasi multikelas pada dataset spiral. Langkah-langkah ini mencakup komputasi forward pass, perhitungan loss, backward pass, dan langkah optimisasi untuk data pelatihan, serta komputasi forward pass dan perhitungan loss untuk data pengujian. Akhirnya, hasilnya dicetak pada setiap iterasi epoch untuk memantau kemajuan pelatihan dan evaluasi model.

```
# Plot decision boundaries for training and test sets on Custom Spiral
Dataset
plt.figure(figsize=(14, 7))
plt.subplot(1, 2, 1)
```

```
plt.title("Decision Boundaries - Custom Spiral Training Data")
plot_decision_boundary_v2(model_custom_spiral, custom_X_train_v2,
custom_y_train_v2)

plt.subplot(1, 2, 2)
plt.title("Decision Boundaries - Custom Spiral Test Data")
plot_decision_boundary_v2(model_custom_spiral, custom_X_test_v2,
custom_y_test_v2)

plt.show()
```

- plt.figure(figsize=(14, 7)): Membuat objek gambar dengan ukuran 14x7 inci untuk menampung dua subplot.
- plt.subplot(1, 2, 1): Membuat subplot pertama dalam grid 1x2 (satu baris, dua kolom) dan fokus pada subplot pertama.
- plt.title("Decision Boundaries - Custom Spiral Training Data"): Memberikan judul pada subplot pertama untuk menunjukkan bahwa ini adalah visualisasi batas keputusan pada data pelatihan.
- plot_decision_boundary_v2(model_custom_spiral, custom_X_train_v2, custom_y_train_v2): Memanggil fungsi plot_decision_boundary_v2 untuk menggambar batas keputusan pada data pelatihan menggunakan model model_custom_spiral.
- plt.subplot(1, 2, 2): Membuat subplot kedua dalam grid 1x2 dan beralih fokus ke subplot kedua.
- plt.title("Decision Boundaries - Custom Spiral Test Data"): Memberikan judul pada subplot kedua untuk menunjukkan bahwa ini adalah visualisasi batas keputusan pada data pengujian.
- plot_decision_boundary_v2(model_custom_spiral, custom_X_test_v2, custom_y_test_v2): Memanggil fungsi plot_decision_boundary_v2 untuk menggambar batas keputusan pada data pengujian menggunakan model yang sama.
- plt.show(): Menampilkan plot keseluruhan dengan dua subplot yang menunjukkan decision boundaries pada data pelatihan dan pengujian dari dataset spiral kustom.

Kode ini bertujuan untuk membuat visualisasi decision boundaries (batas keputusan) pada dataset spiral kustom yang telah dijelaskan sebelumnya untuk memberikan

pemahaman visual tentang bagaimana model yang telah dilatih memisahkan kelas pada dataset spiral kustom. Plot ini membantu dalam mengevaluasi kinerja model pada data yang belum pernah dilihat sebelumnya (data pengujian) dengan melihat sejauh mana model mampu mengeneralisasi pola pada dataset.

Kode di atas mengilustrasikan implementasi lengkap dari pembuatan, pelatihan, dan evaluasi model jaringan saraf untuk tugas klasifikasi pada dua dataset yang berbeda. Pertama, kode memeriksa ketersediaan GPU, membuat dan menginisialisasi model jaringan saraf, dan mengatur data sesuai dengan perangkat yang dipilih. Setelah itu, dilakukan pelatihan model dengan menentukan fungsi kerugian, optimizer, dan melalui iterasi training loop. Loop ini melibatkan langkah-langkah forward pass, perhitungan loss, backward pass, dan langkah optimisasi pada data pelatihan. Selama pelatihan, model dievaluasi pada data pengujian untuk memonitor kinerja dan mencegah overfitting. Selama iterasi pelatihan, metrik akurasi juga dihitung untuk memantau kemajuan model. Kode ini kemudian digunakan untuk membuat visualisasi seperti plot decision boundaries pada dataset dan hasil evaluasi model. Secara keseluruhan, kode mencakup seluruh alur dari awal hingga akhir untuk membangun dan melatih model jaringan saraf pada dua dataset yang berbeda, dengan fokus pada tugas klasifikasi.