Nama : Andhika Aria Pratama N

NIM : 1103202121

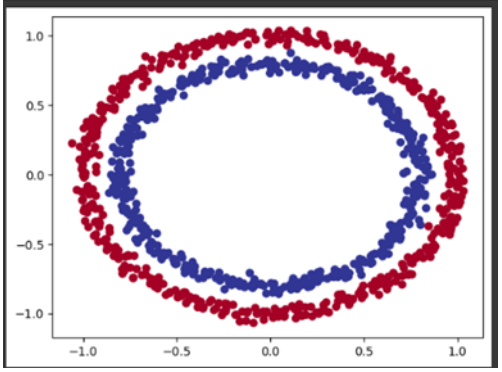
02 PyTorch Neural Network Classification

- Arsitektur klasifikasi neural network

Arsitektur Neural Network Klasifikasi mencerminkan struktur umum dari jaringan saraf yang digunakan untuk tugas klasifikasi. Terdapat hyperparameter yang dapat disesuaikan baik untuk klasifikasi biner maupun klasifikasi banyak kelas. Sebelum membahas penulisan kode, perlu memahami komponen kunci dan hyperparameter yang relevan untuk setiap jenis klasifikasi. Pada klasifikasi biner, hyperparameter meliputi bentuk input layer, konfigurasi hidden layer, jumlah neuron per hidden layer, bentuk output layer, fungsi aktivasi hidden layer, fungsi aktivasi output, fungsi loss (binary crossentropy), dan pilihan optimizer (SGD, Adam, dll.). Untuk klasifikasi banyak kelas, beberapa hyperparameter serupa digunakan, namun terdapat perbedaan dalam output layer shape (sesuai jumlah kelas), fungsi aktivasi output (softmax), dan fungsi loss (cross entropy).

- Membuat klasifikasi data

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




```
# Split 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, # 2
                                                    random_state=42) #


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

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

Langkah awal yaitu memuat library sklearn dengan mengimport lingkaran dengan syntaxnya make_circles. Data tersebut berisi nilai 0 dan 1 serta menghasilkan data yang melingkar serta terdapat 2 value yaitu x dan y. setelahnya data dibuat menjadi 2 variabel x dan y untuk dilakukan train dan test pada kedua data. Masing-masing bobot train dan test data sebesar 80% dan 20%. Bobot tersebut mempunyai data sebesar 800 dan 200 data pada kedua variabel tersebut.

- Membangun model.

```
# Standard PyTorch imports
import torch
from torch import nn

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

```
'cuda'
```

```
# Create a loss function
# loss_fn = nn.BCELoss() # BCELoss = no sigmoid built-in
loss_fn = nn.BCEWithLogitsLoss() # BCEWithLogitsLoss = sigm

# Create an optimizer
optimizer = torch.optim.SGD(params=model_0.parameters(),
                            lr=0.1)
```

```
# Calculate accuracy (a classification metric)
def accuracy_fn(y_true, y_pred):
    correct = torch.eq(y_true, y_pred).sum().item() # torch
    acc = (correct / len(y_pred)) * 100
    return acc
```

langkah pertama melibatkan persiapan kode perangkat-agnostik untuk memastikan kompatibilitas pada CPU atau GPU. Selanjutnya, model dibangun dengan membuat kelas yang mewarisi nn.Module, menggunakan dua lapisan nn.Linear untuk mengelola bentuk input dan output data. Fungsi forward() menentukan langkah maju model, dan model diinstansiasi sebelum dikirimkan ke perangkat target. Penggunaan nn.Sequential diperkenalkan sebagai opsi konstruksi model, tetapi custom nn.Module tetap dibutuhkan untuk kasus yang rumit. Setelah itu, model diuji dengan melakukan prediksi pada data uji dan dievaluasi untuk memastikan keluaran. Pembuatan model klasifikasi PyTorch melibatkan pembuatan model menggunakan PyTorch dengan dua layer linear, menggunakan fungsi aktivasi ReLU di hidden layer dan sigmoid di output layer untuk klasifikasi biner, atau menggunakan nn.Sequential untuk model yang serupa.

- Melatih model

```
# View the frist 5 outputs of the forward pass on the test data
y_logits = model_0(X_test.to(device))[:5]
y_logits
```

```
tensor([[-0.2371],
        [-0.0907],
        [-0.4195],
        [-0.1862],
        [-0.1000]], device='cuda:0', grad_fn=<SliceBackward0>)
```

```
# Use sigmoid on model logits
y_pred_probs = torch.sigmoid(y_logits)
y_pred_probs
```

```
tensor([[0.4410],
        [0.4773],
        [0.3966],
        [0.4536],
        [0.4750]], device='cuda:0', grad_fn=<SigmoidBackward0>)
```

```
# Find the predicted labels (round the prediction probabilities)
y_preds = torch.round(y_pred_probs)

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

# Check for equality
print(torch.eq(y_preds.squeeze(), y_pred_labels.squeeze()))

# Get rid of extra dimension
y_preds.squeeze()
```

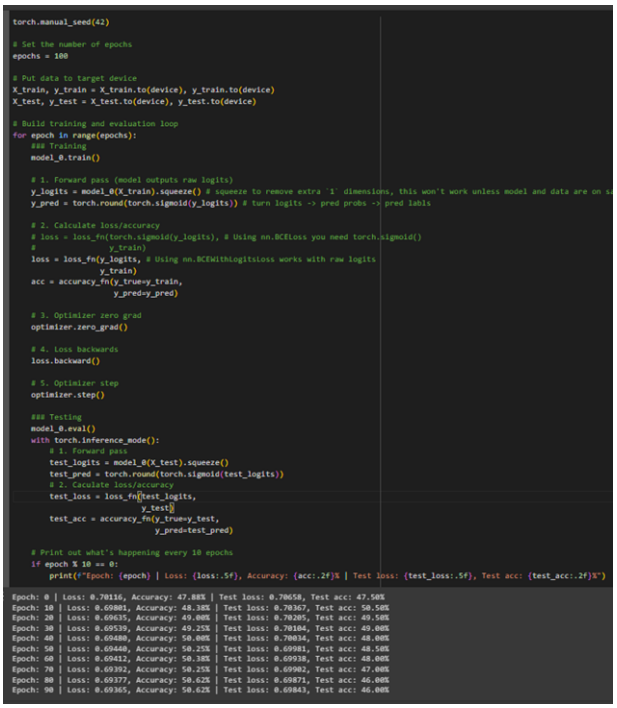
```
tensor([True, True, True, True, True], device='cuda:0')
tensor([0., 0., 0., 0., 0.], device='cuda:0', grad_fn=<SqueezeBackward0>)
```

```
y_test[:5]
```

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

Langkah tersebut mengilustrasikan untuk mendapatkan dan mengevaluasi lima output pertama dari model jaringan saraf pada data uji. Pertama, logit (output mentah model) untuk lima sampel pertama diekstrak, kemudian fungsi aktivasi sigmoid diterapkan untuk mendapatkan prediksi probabilitas. Selanjutnya, probabilitas tersebut dibulatkan untuk mendapatkan label prediksi, kemudian dilakukan pengecekan kesetaraan dengan label sebenarnya, dan hasilnya disusun kembali untuk perbandingan.

Output yang tercetak menunjukkan bahwa label prediksi sesuai dengan label sebenarnya untuk lima sampel pertama pada data uji.



```
torch.manual_seed(42)

# Set the number of epochs
epochs = 100

# Put data to target device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

# Build training and evaluation loop
for epoch in range(epochs):
    ### Training
    model_0.train()

    # 1. Forward pass (model outputs raw logits)
    y_logits = model_0(X_train).squeeze() # squeeze to remove extra `1` dimensions, this won't work unless model and data are on s
    y_pred = torch.round(torch.sigmoid(y_logits)) # turn logits -> pred probs -> pred labls

    # 2. Calculate loss/accuracy
    # loss = loss_fn(torch.sigmoid(y_logits), # Using nn.BCELoss you need torch.sigmoid()
    #                y_train)
    loss = loss_fn(y_logits, # Using nn.BCEWithLogitsLoss works with raw logits
                   y_train)
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_0.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_logits = model_0(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))
        # 2. Caculate loss/accuracy
        test_loss = loss_fn(test_logits,
                            y_test)
        test_acc = accuracy_fn(y_true=y_test,
                               y_pred=test_pred)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% | Test loss: {test_loss:.5f}, Test acc: {test_acc:.2f}%")
```

```
Epoch: 0 | Loss: 0.70116, Accuracy: 47.88% | Test loss: 0.70658, Test acc: 47.50%
Epoch: 10 | Loss: 0.69881, Accuracy: 48.38% | Test loss: 0.70367, Test acc: 50.50%
Epoch: 20 | Loss: 0.69635, Accuracy: 49.00% | Test loss: 0.70205, Test acc: 49.50%
Epoch: 30 | Loss: 0.69539, Accuracy: 49.25% | Test loss: 0.70104, Test acc: 49.00%
Epoch: 40 | Loss: 0.69480, Accuracy: 50.00% | Test loss: 0.70034, Test acc: 48.00%
Epoch: 50 | Loss: 0.69440, Accuracy: 50.25% | Test loss: 0.69981, Test acc: 48.50%
Epoch: 60 | Loss: 0.69412, Accuracy: 50.38% | Test loss: 0.69938, Test acc: 48.00%
Epoch: 70 | Loss: 0.69392, Accuracy: 50.25% | Test loss: 0.69902, Test acc: 47.00%
Epoch: 80 | Loss: 0.69377, Accuracy: 50.62% | Test loss: 0.69871, Test acc: 46.00%
Epoch: 90 | Loss: 0.69365, Accuracy: 50.62% | Test loss: 0.69843, Test acc: 46.00%
```

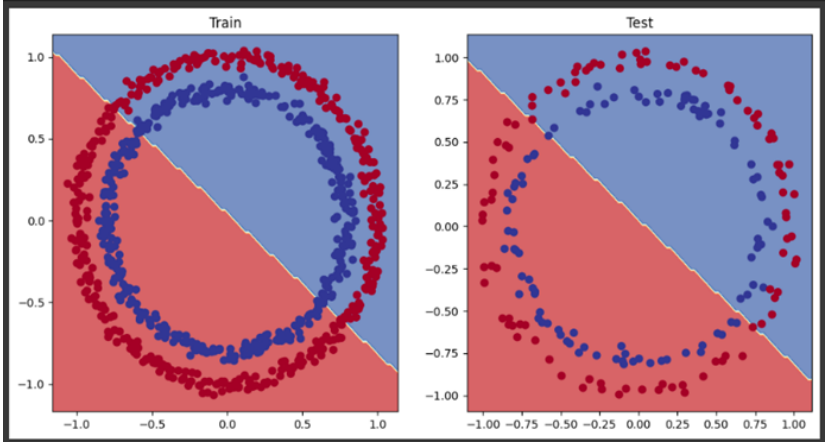
Kode tersebut merupakan implementasi pelatihan dan evaluasi model jaringan saraf menggunakan PyTorch selama 100 epoch dengan data latih, dicetak hasil pelatihan setiap 10 epoch, menampilkan nilai loss, akurasi, test loss, dan test accuracy. Meskipun akurasi pada data uji bervariasi, performa model saat ini stagnan di sekitar 50%, menunjukkan perlu evaluasi lebih lanjut dan penyesuaian untuk meningkatkan kinerja pada tugas klasifikasi.

- Membuat prediksi dan evaluasi model

```
import requests
from pathlib import Path

# Download helper functions from Learn PyTorch repo (if not already downloaded)
if Path("helper_functions.py").is_file():
  print("helper_functions.py already exists, skipping download")
else:
  print("Downloading helper_functions.py")
  request = requests.get("https://raw.githubusercontent.com/mrdbourke/pytorch-deep-learning/main/helper_functions.py")
  with open("helper_functions.py", "wb") as f:
    f.write(request.content)

from helper_functions import plot_predictions, plot_decision_boundary
```

```
Downloading helper_functions.py
```

```
# 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)
```



Dalam upaya pemahaman lebih lanjut, dilakukan visualisasi menggunakan decision boundary sebagai batas pemisah antara kelas yang dihasilkan oleh model. Plot tersebut menunjukkan bahwa model saat ini mencoba memisahkan data dengan garis lurus, yang tidak sesuai dengan distribusi data berbentuk lingkaran, dan hal ini menjelaskan performa rendah model karena garis lurus hanya mampu memotong data menjadi dua bagian sejajar tanpa mampu menangkap pola yang sesuai. Identifikasi kondisi ini sebagai underfitting dalam terminologi machine learning, di mana model tidak mampu menangkap pola yang ada dalam data, sehingga tidak dapat membuat prediksi yang akurat.

- Meningkatkan model

```python
torch.manual_seed(42)

epochs = 1000 # Train for longer

# Put data to target device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

for epoch in range(epochs):
    ### Training
    # 1. Forward pass
    y_logits = model_1(X_train).squeeze()
    y_pred = torch.round(torch.sigmoid(y_logits)) # logits -> predicition probabilit

    # 2. Calculate loss/accuracy
    loss = loss_fn(y_logits, y_train)
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_1.eval()
    with torch.inference_mode():
        # 1. Forward pass
        test_logits = model_1(X_test).squeeze()
        test_pred = torch.round(torch.sigmoid(test_logits))
        # 2. Caculate loss/accuracy
        test_loss = loss_fn(test_logits,
                            y_test)
        test_acc = accuracy_fn(y_true=y_test,
                               y_pred=test_pred)

    # Print out what's happening every 10 epochs
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% | Test loss:
```

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

Dalam upaya meningkatkan model dari perspektif model itu sendiri, beberapa pendekatan dapat diterapkan, seperti menambah lapisan atau unit tersembunyi, meningkatkan jumlah epochs, mengubah fungsi aktivasi, menyesuaikan learning rate, dan merubah fungsi loss. Penerapan perbaikan pada model dilakukan dengan menambah lapisan dan unit tersembunyi, serta melatih untuk lebih lama (epochs=1000) pada CircleModelV1. Meskipun arsitektur model lebih kompleks, setelah pelatihan lebih lama, model ini belum menunjukkan peningkatan yang signifikan dalam pembelajaran pola dari data, yang memicu eksplorasi lebih lanjut terhadap masalah tersebut.

- Satuan yang hilang (non linear)

```python
# Fit the model
torch.manual_seed(42)
epochs = 1000

# Put all data on target device
X_train, y_train = X_train.to(device), y_train.to(device)
X_test, y_test = X_test.to(device), y_test.to(device)

for epoch in range(epochs):
    # 1. Forward pass
    y_logits = model_3(X_train).squeeze()
    y_pred = torch.round(torch.sigmoid(y_logits)) # logits -> prediction probabilities -> prediction labels

    # 2. Calculate loss and accuracy
    loss = loss_fn(y_logits, y_train) # BCEWithLogitsLoss calculates loss using logits
    acc = accuracy_fn(y_true=y_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_3.eval()
    with torch.inference_mode():
      # 1. Forward pass
      test_logits = model_3(X_test).squeeze()
      test_pred = torch.round(torch.sigmoid(test_logits)) # logits -> prediction probabilities -> prediction labels
      # 2. Calcuate loss and accuracy
      test_loss = loss_fn(test_logits, y_test)
      test_acc = accuracy_fn(y_true=y_test,
                             y_pred=test_pred)

    # Print out what's happening
    if epoch % 100 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Accuracy: {acc:.2f}% | Test Loss: {test_loss:.5f}, Test Accuracy: {test_acc:.2f}%")
```

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

Dalam mengatasi keterbatasan model untuk menggambar garis lurus, kita mempertimbangkan solusi dengan memberikan kapasitas pada model untuk menggambar garis yang tidak lurus (non-linear). Untuk mengimplementasikan hal ini, kita membangun model dengan menambahkan fungsi aktivasi non-linear, seperti ReLU (Rectified Linear Unit), di antara lapisan tersembunyi. ReLU membantu model memahami pola non-linear dari data dan melalui pelatihan dengan dataset baru yang mencakup lingkaran merah dan biru, kita dapat memantau bagaimana model mampu mengatasi pola-pola tersebut serta mengevaluasi performanya pada data uji. Hasil evaluasi nantinya dapat dibandingkan dengan model sebelumnya yang hanya menggunakan linearitas, memungkinkan kita untuk menilai dampak dari penambahan non-linearitas terhadap kemampuan model. Dihasilkan akurasi yang didapatkan meningkat hingga 74% lebih baik dari pada sebelumnya.

- Mereplikasi fungsi aktivasi non-linier

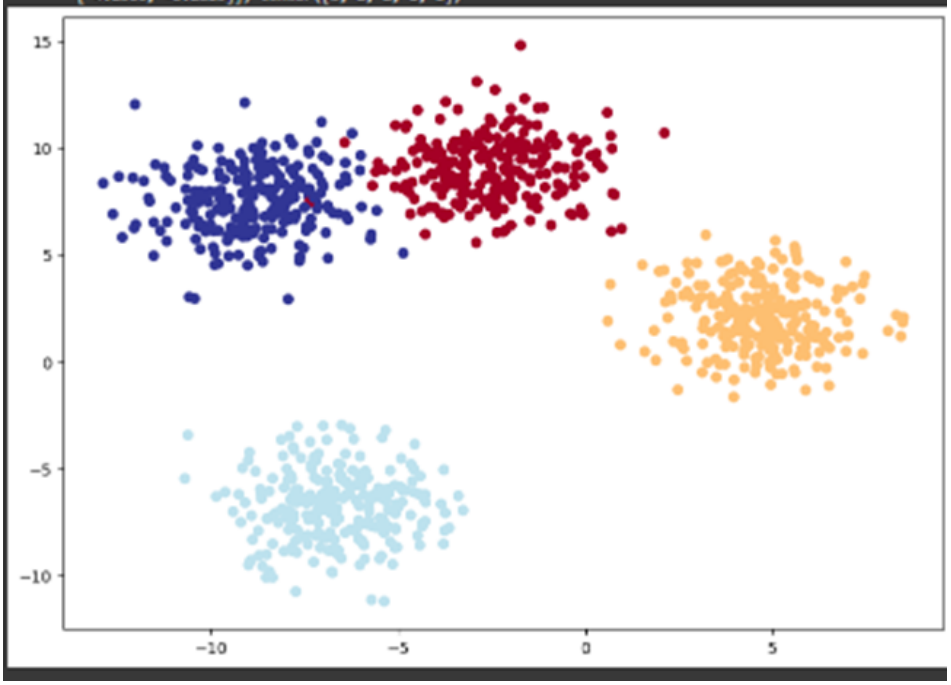
Setelah mengeksplorasi peningkatan model dengan menambahkan fungsi aktivasi non-linear, langkah selanjutnya adalah mereplikasi fungsi-fungsi tersebut untuk lebih memahami peran mereka. Dalam konteks ini, kita akan terlebih dahulu membuat data kecil menggunakan tensor PyTorch. Fokus pertama adalah mereplikasi fungsi ReLU (Rectified Linear Unit), yang mengubah nilai negatif menjadi 0 dan membiarkan nilai positif tidak berubah. Selanjutnya, juga mencoba mengimplementasikan fungsi sigmoid, yang digunakan sebelumnya, untuk menghasilkan nilai antara 0 dan 1. Setelah mereplikasi kedua fungsi ini, hasilnya akan divisisalisasikan untuk memvisualisasikan perbedaan antara garis lurus awal dan garis yang diubah oleh fungsi aktivasi non-linear.

- Menyatukan semuanya dengan membangun model PyTorch multikelas

- Membuat data klasifikasi kelas jamak

```python
# Import dependencies
import torch
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split

# Set the hyperparameters for data creation
NUM_CLASSES = 4
NUM_FEATURES = 2
RANDOM_SEED = 42

# 1. Create multi-class data
X_blob, y_blob = make_blobs(n_samples=1000,
    n_features=NUM_FEATURES, # X features
    centers=NUM_CLASSES, # y labels
    cluster_std=1.5, # give the clusters a little shake up (try changing this to 1.0, the default)
    random_state=RANDOM_SEED
)

# 2. Turn data into tensors
X_blob = torch.from_numpy(X_blob).type(torch.float)
y_blob = torch.from_numpy(y_blob).type(torch.LongTensor)
print(X_blob[:5], y_blob[:5])

# 3. Split into train and test sets
X_blob_train, X_blob_test, y_blob_train, y_blob_test = train_test_split(X_blob,
    y_blob,
    test_size=0.2,
    random_state=RANDOM_SEED
)

# 4. Plot data
plt.figure(figsize=(10, 7))
plt.scatter(X_blob[:, 0], X_blob[:, 1], c=y_blob, cmap=plt.cm.RdYlBu);
```

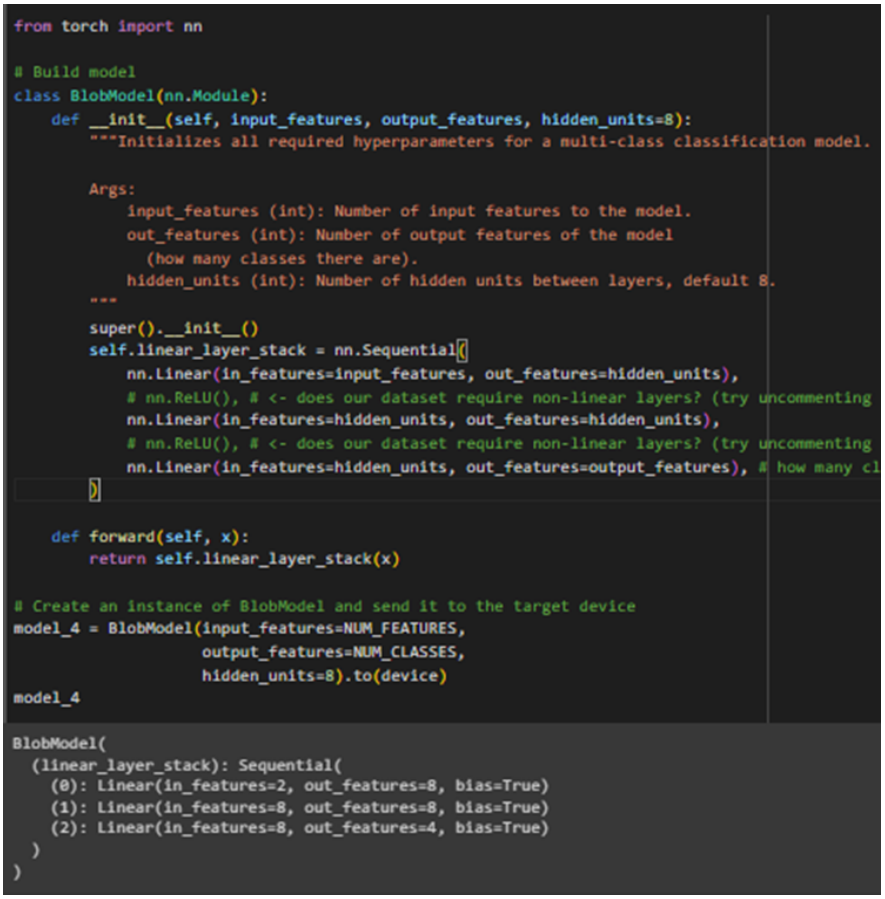
```
tensor([[-8.4134,  6.9352],
        [-5.7665, -6.4312],
        [-6.0421, -6.7661],
        [ 3.9588,  0.6984],
        [ 4.2505, -0.2815]]) tensor([3, 2, 2, 1, 1])
```



Kode tersebut menciptakan dan memvisualisasikan dataset multi-kelas menggunakan fungsi `make_blobs` dari scikit-learn, kemudian mengonversinya menjadi tensor PyTorch. Visualisasi

menunjukkan sebaran titik-titik data berwarna yang membedakan antar-kelas di dalam cluster-cluster pada bidang dua dimensi.

- Membangun model klasifikasi kelas jamak di PyTorch

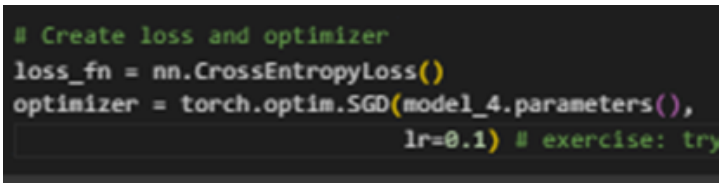


```
from torch import nn

# Build model
class BlobModel(nn.Module):
    def __init__(self, input_features, output_features, hidden_units=8):
        """Initializes all required hyperparameters for a multi-class classification model.

        Args:
            input_features (int): Number of input features to the model.
            out_features (int): Number of output features of the model
              (how many classes there are).
            hidden_units (int): Number of hidden units between layers, default 8.
        """
        super().__init__()
        self.linear_layer_stack = nn.Sequential(
            nn.Linear(in_features=input_features, out_features=hidden_units),
            # nn.ReLU(), # <- does our dataset require non-linear layers? (try uncommenting
            nn.Linear(in_features=hidden_units, out_features=hidden_units),
            # nn.ReLU(), # <- does our dataset require non-linear layers? (try uncommenting
            nn.Linear(in_features=hidden_units, out_features=output_features), # how many cl
        )

    def forward(self, x):
        return self.linear_layer_stack(x)

# Create an instance of BlobModel and send it to the target device
model_4 = BlobModel(input_features=NUM_FEATURES,
                    output_features=NUM_CLASSES,
                    hidden_units=8).to(device)
model_4
```

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

Kode tersebut mendefinisikan model `BlobModel` untuk klasifikasi multi-kelas dengan lapisan-lapisan linear. Model ini memiliki tiga lapisan linear, yang dua di antaranya dapat diaktifkan dengan fungsi ReLU. Model ini dibuat dengan jumlah input features sebanyak 2, hidden units sebanyak 8, dan output features sebanyak 4 (sesuai dengan jumlah kelas). Hasilnya adalah model dengan tiga lapisan linear berurutan.

- Membuat fungsi kerugian dan pengoptimal untuk model PyTorch multikelas



```
# Create loss and optimizer
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model_4.parameters(),
                            lr=0.1) # exercise: try
```

Bertujuan untuk membuat fungsi kerugian (loss function) dan pengoptimal (optimizer) yang akan digunakan dalam pelatihan model `BlobModel`. Fungsi kerugian yang digunakan adalah `nn.CrossEntropyLoss()`, yang umum digunakan untuk tugas klasifikasi multi-kelas. Pengoptimal yang digunakan adalah Stochastic Gradient Descent (SGD) dengan laju pembelajaran (learning rate)

0.1. Pengguna diberikan opsi untuk mencoba mengubah laju pembelajaran dan melihat bagaimana performa model berubah sebagai hasil dari eksperimen tersebut.

- Mendapatkan probabilitas prediksi untuk model PyTorch multikelas

```
# Perform a single forward pass on the data (we'll need to
model_4(X_blob_train.to(device))[:5]
```

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

```
# How many elements in a single prediction sample?
model_4(X_blob_train.to(device))[0].shape, NUM_CLASSES
```

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

```
# Make prediction logits with model
y_logits = model_4(X_blob_test.to(device))

# Perform softmax calculation on logits across dimension 1
y_pred_probs = torch.softmax(y_logits, dim=1)
print(y_logits[:5])
print(y_pred_probs[:5])
```

```
tensor([[-1.2549, -0.8112, -1.4795, -0.5696],
        [ 1.7168, -1.2270,  1.7367,  2.1010],
        [ 2.2400,  0.7714,  2.6020,  1.0107],
        [-0.7993, -0.3723, -0.9138, -0.5388],
        [-0.4332, -1.6117, -0.6891,  0.6852]], device='cuda
       grad_fn=<SliceBackward0>)
tensor([[0.1872, 0.2918, 0.1495, 0.3715],
        [0.2824, 0.0149, 0.2881, 0.4147],
        [0.3380, 0.0778, 0.4854, 0.0989],
        [0.2118, 0.3246, 0.1889, 0.2748],
        [0.1945, 0.0598, 0.1506, 0.5951]], device='cuda:0',
       grad_fn=<SliceBackward0>)
```

```
# Sum the first sample output of the softmax activation fu
torch.sum(y_pred_probs[0])
```

```
tensor(1., device='cuda:0', grad_fn=<SumBackward0>)
```

```
# Which class does the model think is *most* likely at the
print(y_pred_probs[0])
print(torch.argmax(y_pred_probs[0]))
```

```
tensor([0.1872, 0.2918, 0.1495, 0.3715], device='cuda:0',
       grad_fn=<SelectBackward0>)
tensor(3, device='cuda:0')
```

Secara keseluruhan, kode tersebut adalah implementasi pembuatan, pelatihan, dan evaluasi model klasifikasi multikelas menggunakan PyTorch. Langkah-langkahnya melibatkan pembuatan dataset sintetis dengan beberapa kelas menggunakan fungsi `make_blobs` dari scikit-learn, konversi dataset

ke dalam format tensor PyTorch, dan pembagian data menjadi set latih dan uji. Model klasifikasi multikelas yang disebut `BlobModel` kemudian dibuat, memiliki beberapa lapisan linear. Fungsi kerugian CrossEntropy dan pengoptimal SGD diinisialisasi untuk pelatihan, dan model dilatih dengan data latih. Selanjutnya, dilakukan pengujian model pada data uji, dengan perhitungan logits, softmax, dan argmax untuk mendapatkan prediksi dalam bentuk probabilitas kelas. Melalui langkah-langkah ini, pengguna juga diminta untuk mencoba variasi laju pembelajaran guna mengamati pengaruhnya terhadap performa model.

- Membuat loop pelatihan dan pengujian untuk model PyTorch multikelas

```
# Fit the model
torch.manual_seed(42)

# Set number of epochs
epochs = 100

# Put data to target device
X_blob_train, y_blob_train = X_blob_train.to(device), y_blob_train.to(device)
X_blob_test, y_blob_test = X_blob_test.to(device), y_blob_test.to(device)

for epoch in range(epochs):
    ### Training
    model_4.train()

    # 1. Forward pass
    y_logits = model_4(X_blob_train) # model outputs raw logits
    y_pred = torch.softmax(y_logits, dim=1).argmax(dim=1) # go from logits -> prediction probabilities -> prediction labels
    # print(y_logits)
    # 2. Calculate loss and accuracy
    loss = loss_fn(y_logits, y_blob_train)
    acc = accuracy_fn(y_true=y_blob_train,
                      y_pred=y_pred)

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backwards
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    ### Testing
    model_4.eval()
    with torch.inference_mode():
      # 1. Forward pass
      test_logits = model_4(X_blob_test)
      test_pred = torch.softmax(test_logits, dim=1).argmax(dim=1)
      # 2. Calculate test loss and accuracy
      test_loss = loss_fn(test_logits, y_blob_test)
      test_acc = accuracy_fn(y_true=y_blob_test,
                             y_pred=test_pred)

    # Print out what's happening
    if epoch % 10 == 0:
        print(f"Epoch: {epoch} | Loss: {loss:.5f}, Acc: {acc:.2f}% | Test Loss: {test_loss:.5f}, Test Acc: {test_acc:.2f}%")
```

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

Kode tersebut merupakan proses pelatihan dan evaluasi model klasifikasi multikelas menggunakan model `BlobModel`. Model dilatih selama 100 epoch dengan data latih, dan hasil pelatihan dicetak setiap 10 epoch. Hasil output mencakup nilai loss, akurasi pada data latih, test loss, dan test accuracy pada setiap checkpoint epoch. Performa model terlihat sangat baik dan meningkat secara signifikan dari sebelumnya, dengan akurasi pada data uji mencapai 99.50% setelah beberapa epoch, menunjukkan bahwa model berhasil memahami dan menggeneralisasi pola pada data multikelas yang berbentuk lingkaran.

- Membuat dan mengevaluasi prediksi dengan model kelas jamak PyTorch

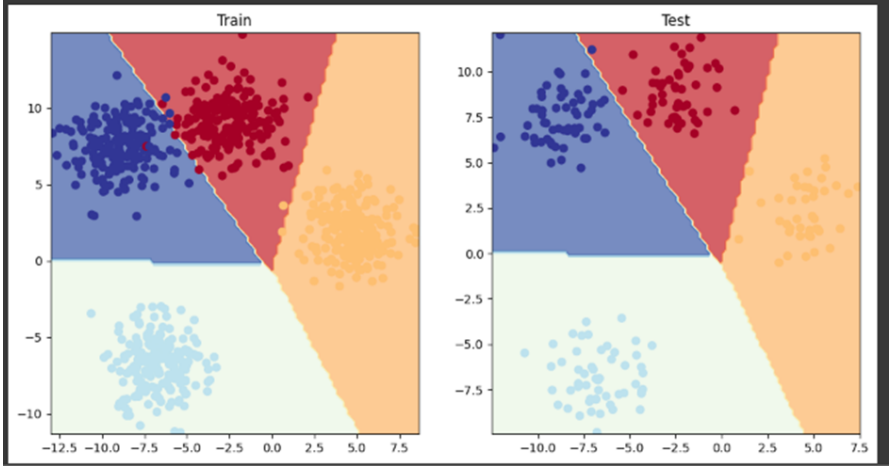
```python
# Make predictions
model_4.eval()
with torch.inference_mode():
    y_logits = model_4(X_blob_test)

# View the first 10 predictions
y_logits[:10]
```

```
tensor([[  4.3377,  10.3539, -14.8948,  -9.7642],
        [  5.0142, -12.0371,   3.3860,  10.6699],
        [ -5.5885, -13.3448,  20.9894,  12.7711],
        [  1.8400,   7.5599,  -8.6016,  -6.9942],
        [  8.0726,   3.2906, -14.5998,  -3.6186],
        [  5.5844, -14.9521,   5.0168,  13.2890],
        [ -5.9739, -10.1913,  18.8655,   9.9179],
        [  7.0755,  -0.7601,  -9.5531,   0.1736],
        [ -5.5918, -18.5990,  25.5309,  17.5799],
        [  7.3142,   0.7197, -11.2017,  -1.2011]], device='cuda:0')
```

```python
# Turn predicted logits in prediction probabilities
y_pred_probs = torch.softmax(y_logits, dim=1)

# Turn prediction probabilities into prediction labels
y_preds = y_pred_probs.argmax(dim=1)

# Compare first 10 model preds and test labels
print(f"Predictions: {y_preds[:10]}\nLabels: {y_blob_test[:10]}")
print(f"Test accuracy: {accuracy_fn(y_true=y_blob_test, y_pred=y_preds)}%")
```

```
Predictions: tensor([1, 3, 2, 1, 0, 3, 2, 0, 2, 0], device='cuda:0')
Labels: tensor([1, 3, 2, 1, 0, 3, 2, 0, 2, 0], device='cuda:0')
Test accuracy: 99.5%
```

```python
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Train")
plot_decision_boundary(model_4, X_blob_train, y_blob_train)
plt.subplot(1, 2, 2)
plt.title("Test")
plot_decision_boundary(model_4, X_blob_test, y_blob_test)
```



Pada tahap ini, logits hasil prediksi dari model diubah menjadi probabilitas menggunakan fungsi softmax, lalu probabilitas tersebut dijadikan label prediksi dengan mengambil indeks dengan probabilitas tertinggi. Hasil prediksi kemudian dibandingkan dengan label sebenarnya pada data uji. Dalam contoh ini, 10 prediksi pertama model dan label data uji ditampilkan, dan akurasi pengujian dicetak, yang mencapai tingkat akurasi 99.5%. Hasil ini menunjukkan bahwa model dengan baik memprediksi kelas data uji dengan tingkat akurasi yang tinggi.

- Metrik evaluasi klasifikasi lainnya

Penilaian model klasifikasi melibatkan beberapa metrik evaluasi untuk memberikan pemahaman yang lebih lengkap tentang kinerja model. Selain akurasi (Accuracy) dan loss, terdapat metrik lain yang sering digunakan:

1. Presisi (Precision): Merupakan proporsi dari benar positif dibandingkan dengan total prediksi positif. Presisi yang lebih tinggi menghasilkan lebih sedikit positif palsu, di mana model memprediksi 1 ketika seharusnya 0. Formulanya adalah TP / (TP + FP).

2. Recall: Merupakan proporsi dari benar positif dibandingkan dengan total benar positif dan falsa negatif. Recall yang lebih tinggi menghasilkan lebih sedikit negatif palsu. Formulanya adalah TP / (TP + FN).

3. F1-score: Menggabungkan presisi dan recall menjadi satu metrik. Skor F1 yang tinggi menunjukkan keseimbangan yang baik antara presisi dan recall. Formulanya adalah 2 * (Presisi * Recall) / (Presisi + Recall).

4. Confusion Matrix: Membandingkan nilai prediksi dengan nilai sebenarnya dalam cara tabel, di mana jika 100% benar, semua nilai dalam matriks akan berada di sepanjang diagonal. Ini memberikan wawasan tentang kinerja model untuk setiap kelas.

5. Classification Report: Kumpulan beberapa metrik klasifikasi utama seperti presisi, recall, dan F1-score. Ini memberikan pandangan holistik tentang kinerja model untuk setiap kelas.

Untuk mengimplementasikan metrik-metrik ini dalam PyTorch, Anda dapat menggunakan pustaka TorchMetrics. Jika Anda lebih suka menggunakan implementasi Scikit-Learn, metode yang sesuai tersedia dalam pustaka tersebut. Metrik-metrik ini membantu menyediakan analisis yang lebih mendalam tentang seberapa baik model klasifikasi berkinerja di luar akurasi saja.