Nama: Andi Cleopatra Maryam jamila

Nim: 1103213071

Week 11 Deep learning model

```
# Import library
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
# Load dataset
df = pd.read_csv('/content/sample_data/heart.csv')

# Menampilkan beberapa baris pertama untuk melihat struktur data
df.head()
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Analisis:

Mengimpor pustaka yang diperlukan untuk membangun dan melatih model deep learning menggunakan PyTorch, termasuk pustaka untuk memanipulasi dataset, seperti pandas untuk memuat dan memanipulasi data, serta sklearn untuk normalisasi dan pembagian data. Dataset yang digunakan adalah 'heart.csv', yang dimuat dengan pd.read_csv(). Kemudian, df.head() digunakan untuk menampilkan beberapa baris pertama dari dataset, memungkinkan pengguna untuk memeriksa struktur data dan memahami fitur serta target yang ada. Tujuan dari kode ini adalah untuk mempersiapkan data sebelum diproses lebih lanjut, seperti normalisasi dan pembagian data menjadi set pelatihan dan pengujian, untuk digunakan dalam model klasifikasi.

```
# Pisahkan fitur dan target
X = df.drop('target', axis=1).values  # Semua kolom kecuali 'target' sebagai fitur
y = df['target'].values  # Kolom 'target' sebagai label
```

```
# Normalisasi fitur
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# Split data menjadi train dan test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Konversi ke tensor PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)  # target harus dalam bentuk long
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

Analisis:

Mempersiapkan data untuk model deep learning dengan memisahkan fitur dan target dari dataset. Fitur disimpan dalam variabel X dengan menghapus kolom 'target', sementara label atau target disimpan dalam variabel y. Selanjutnya, fitur dinormalisasi menggunakan StandardScaler untuk memastikan bahwa setiap fitur memiliki distribusi yang seragam, yang penting untuk mempercepat konvergensi model. Data kemudian dibagi menjadi set pelatihan dan pengujian dengan proporsi 80:20 menggunakan train_test_split. Setelah itu, data diubah menjadi tensor PyTorch agar bisa digunakan dalam pelatihan model, dengan memastikan bahwa label (y_train dan y_test) berada dalam format long, yang diperlukan untuk klasifikasi dalam PyTorch. Langkah-langkah ini memastikan bahwa data siap untuk proses pelatihan model deep learning.

```
# Fungsi untuk membuat model MLP dengan parameter yang dapat disesuaikan
class MLPModel(nn.Module):
    def __init__(self, input_dim, hidden_layers, hidden_units, activation_function):
        super(MLPModel, self).__init__()

        # Menentukan layers berdasarkan konfigurasi
        layers = []
        prev_units = input_dim
        for _ in range(hidden_layers):
            layers.append(nn.Linear(prev_units, hidden_units))  # Menambahkan layer linear
            if activation_function == 'ReLU':
                layers.append(nn.ReLU())  # Fungsi aktivasi ReLU
            elif activation_function == 'Sigmoid':
                layers.append(nn.Sigmoid())  # Fungsi aktivasi Sigmoid
            elif activation_function == 'Tanh':
                layers.append(nn.Tanh())  # Fungsi aktivasi Tanh
            elif activation_function == 'Softmax':
                layers.append(nn.Softmax(dim=1))  # Fungsi aktivasi Softmax
            prev_units = hidden_units

        layers.append(nn.Linear(prev_units, 2))  # Output layer untuk klasifikasi 2 kelas
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)
```

Analisis:

Fungsi MLPModel mendefinisikan sebuah model jaringan saraf multi-layer perceptron (MLP) yang fleksibel dan dapat disesuaikan berdasarkan beberapa parameter. Dalam konstruktor __init__, model ini menerima empat parameter utama: input_dim untuk dimensi input (jumlah fitur), hidden_layers untuk menentukan jumlah lapisan tersembunyi, hidden_units untuk jumlah neuron di setiap lapisan tersembunyi, dan activation_function untuk memilih fungsi aktivasi yang akan digunakan di setiap lapisan. Model ini pertama-tama menambahkan lapisan linear (nn.Linear) dengan jumlah unit yang ditentukan, kemudian menambahkan fungsi aktivasi sesuai dengan parameter yang dipilih, seperti ReLU, Sigmoid, Tanh, atau Softmax. Proses ini dilakukan berulang sesuai dengan jumlah lapisan tersembunyi yang diinginkan. Setelah lapisan tersembunyi, lapisan output ditambahkan dengan dua unit (karena model ini untuk klasifikasi dua kelas). Fungsi forward kemudian mendefinisikan alur data melalui jaringan, mengirimkan input melalui model yang sudah dibangun.

```python
# Parameter eksperimen
epochs = [1, 10, 25, 50, 100, 250]
learning_rates = [10, 1, 0.1, 0.01, 0.001, 0.0001]
batch_sizes = [16, 32, 64, 128, 256, 512]
hidden_layers = [1, 2, 3]
hidden_units = [4, 8, 16, 32, 64]
activation_functions = ['ReLU', 'Sigmoid', 'Tanh', 'Softmax']
```

```python
# Fungsi untuk melatih model
def train_model(X_train_tensor, y_train_tensor, epochs, learning_rate, batch_size, hidden_layers, hidden_units, activation_function):
    # Membuat DataLoader untuk batch training
    dataset = TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

    # Inisialisasi model
    input_dim = X_train_tensor.shape[1]
    model = MLPModel(input_dim, hidden_layers, hidden_units, activation_function)

    # Inisialisasi loss function dan optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Melatih model
    for epoch in range(epochs):
        model.train()
        for data, target in train_loader:
            optimizer.zero_grad()  # Reset gradien
            output = model(data)  # Forward pass
            loss = criterion(output, target)  # Hitung loss
            loss.backward()  # Backward pass
            optimizer.step()  # Update parameter model

    return model
```

```
(parameter) shuffle: bool | None
```

```python
# Menguji model dengan berbagai konfigurasi
best_model = None
best_accuracy = 0

for epoch in epochs:
    for lr in learning_rates:
        for batch_size in batch_sizes:
            for hl in hidden_layers:
                for hu in hidden_units:
                    for af in activation_functions:
                        model = train_model(X_train_tensor, y_train_tensor, epoch, lr, batch_size, hl, hu, af)
                        # Evaluasi akurasi model (di sini hanya contoh evaluasi dengan akurasi)
                        model.eval()
                        with torch.no_grad():
                            outputs = model(X_test_tensor)
                            _, predicted = torch.max(outputs, 1)
                            accuracy = (predicted == y_test_tensor).sum().item() / y_test_tensor.size(0)
                            if accuracy > best_accuracy:
                                best_accuracy = accuracy
                                best_model = model
                        print(f"Epoch: {epoch}, LR: {lr}, Batch Size: {batch_size}, Hidden Layers: {hl}, Units: {hu}, Activation: {af}, Accura
```

Output:

```
Epoch: 1, LR: 0.01, Batch Size: 512, Hidden Layers: 3, Units: 32, Activation: Softmax, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.01, Batch Size: 512, Hidden Layers: 3, Units: 64, Activation: ReLU, Accuracy: 0.4975609756097561
Epoch: 1, LR: 0.01, Batch Size: 512, Hidden Layers: 3, Units: 64, Activation: Sigmoid, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.01, Batch Size: 512, Hidden Layers: 3, Units: 64, Activation: Tanh, Accuracy: 0.5414634146341464
Epoch: 1, LR: 0.01, Batch Size: 512, Hidden Layers: 3, Units: 64, Activation: Softmax, Accuracy: 0.4975609756097561
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: ReLU, Accuracy: 0.4975609756097561
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Sigmoid, Accuracy: 0.5073170731707317
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Tanh, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Softmax, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: ReLU, Accuracy: 0.4926829268292683
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Sigmoid, Accuracy: 0.5365853658536586
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Tanh, Accuracy: 0.6390243902439025
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Softmax, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: ReLU, Accuracy: 0.48292682926829267
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Sigmoid, Accuracy: 0.5219512195121951
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Tanh, Accuracy: 0.6585365853658537
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Softmax, Accuracy: 0.4975609756097561
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: ReLU, Accuracy: 0.5219512195121951
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Sigmoid, Accuracy: 0.5560975609756098
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Tanh, Accuracy: 0.6
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Softmax, Accuracy: 0.5024390243902439
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: ReLU, Accuracy: 0.5609756097560976
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Sigmoid, Accuracy: 0.5804878048780487
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Tanh, Accuracy: 0.5707317073170731
Epoch: 1, LR: 0.001, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Softmax, Accuracy: 0.551219512195122
```

Analisis:

Melakukan eksperimen untuk melatih dan menguji model Multi-Layer Perceptron (MLP) dengan berbagai konfigurasi parameter untuk menemukan kombinasi terbaik berdasarkan akurasi. Beberapa parameter eksperimen yang diuji meliputi jumlah epoch (1, 10, 25, 50, 100, 250), learning rate (10, 1, 0.1, 0.01, 0.001, 0.0001), batch size (16, 32, 64, 128, 256, 512), jumlah hidden layers (1, 2, 3), jumlah unit di setiap hidden layer (4, 8, 16, 32, 64), dan fungsi aktivasi (ReLU, Sigmoid, Tanh, Softmax). Fungsi train_model berfungsi untuk melatih model berdasarkan kombinasi parameter yang diberikan, sementara bagian utama eksperimen melakukan pelatihan model dengan konfigurasi yang berbeda dan mengukur akurasi model pada data test. Akurasi dihitung dengan membandingkan prediksi model dengan label yang benar. Setiap kombinasi parameter dicetak dengan nilai akurasinya, dan model dengan akurasi tertinggi disimpan sebagai model terbaik. Dengan cara ini, eksperimen ini bertujuan untuk mengevaluasi dan menemukan kombinasi parameter yang menghasilkan model dengan akurasi terbaik, yang kemudian dapat digunakan untuk tugas klasifikasi di masa depan.

```
# Menganalisis hasil terbaik
print(f"Best Model Accuracy: {best_accuracy}")
```

Output:

```
Best Model Accuracy: 0.9853658536585366
```

Analisis:

Kode print(f'Best Model Accuracy: {best_accuracy}") digunakan untuk menampilkan nilai akurasi terbaik dari model yang telah dievaluasi. Kode ini memanfaatkan *formatted string literals* (*f-strings*) untuk mencetak teks bersama dengan nilai variabel best_accuracy. Variabel ini diharapkan sudah didefinisikan sebelumnya dan berisi nilai akurasi model terbaik dalam format numerik. Outputnya memberikan informasi ringkas kepada pengguna mengenai performa tertinggi model yang diuji.

Misalnya, jika best_accuracy = 0.85, maka hasilnya akan tercetak sebagai "Best Model Accuracy: 0.85".

```
# Menyimpan model terbaik
torch.save(best_model.state_dict(), 'best_heart_model.pth')
```

Analisis:

Kode torch.save(best_model.state_dict(), 'best_heart_model.pth') digunakan untuk menyimpan parameter (bobot dan bias) dari model pembelajaran mesin best_model ke dalam file bernama 'best_heart_model.pth'. Dengan menggunakan fungsi state_dict() dari PyTorch, kode ini memastikan hanya parameter model yang disimpan, bukan seluruh objek model, sehingga lebih ringan dan fleksibel untuk digunakan kembali. File hasil penyimpanan dapat digunakan untuk memuat kembali model pada sesi lain tanpa perlu melatih ulang, yang sangat berguna dalam pengembangan dan implementasi model pembelajaran mesin.