Nama : Ameliani Kusmayadi
NIM : 1103213044
Kelas : TK-45-06

WEEK 11 MLP DEEP LEARNING

```
# 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 the pandas library and assign it to the variable '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/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

Penjelasan:
Output yang ditampilkan adalah beberapa baris pertama dari DataFrame df. Setiap baris mewakili satu data point atau satu pasien dalam dataset penyakit jantung. Setiap kolom mewakili satu fitur atau atribut dari pasien tersebut

```
# 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
```

Penjelasan:
Kode Python di atas bertujuan untuk memisahkan dataset menjadi dua bagian utama sebelum dilakukan pemodelan.

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

Penjelasan:
Normalisasi fitur dengan menggunakan StandardScaler adalah langkah penting dalam pra-pemrosesan data untuk meningkatkan kinerja model machine learning. Dengan memahami konsep ini, membuat model yang lebih baik dan lebih akurat.

```python
# 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)
```

Penjelasan:
Kode ini digunakan untuk membagi dataset menjadi Membagi Data Menjadi Train dan Test Set.

```python
# 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)
```

Penjelasan:
Bagian ini mengubah data dari format NumPy array menjadi tensor PyTorch, yang merupakan struktur data dasar dalam PyTorch. Tensor ini akan digunakan untuk melakukan komputasi dalam deep learning.

```python
# 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)
```

Penjelasan:
Kode ini memberikan fleksibilitas dalam membangun berbagai jenis arsitektur MLP dengan mengatur parameter seperti jumlah hidden layer, jumlah unit pada setiap layer, dan jenis fungsi aktivasi. Setelah objek MLPModel dibuat, dapat melatihnya dengan data pelatihan dan menggunakannya untuk membuat prediksi pada data baru.

```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']
```

Penjelasan:

Kode di atas memberikan kerangka kerja untuk melakukan eksperimen yang komprehensif untuk menemukan konfigurasi terbaik untuk model MLP.

```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
```

Penjelasan:

Kode ini secara keseluruhan mengimplementasikan proses pelatihan dasar untuk jaringan saraf tiruan. Parameter-parameter yang dapat disesuaikan (epochs, learning rate, batch size, dll.) memungkinkan kita untuk melakukan eksperimen dan mencari konfigurasi terbaik.

```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}, Accuracy: {accuracy}")
```

Output:

```
Output streaming akan dipotong hingga 5000 baris terakhir.
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 2, Units: 64, Activation: Tanh, Accuracy: 0.9853658536585366
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 2, Units: 64, Activation: Softmax, Accuracy: 0.8926829268292
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 4, Activation: ReLU, Accuracy: 0.848780487804878
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 4, Activation: Sigmoid, Accuracy: 0.8
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 4, Activation: Tanh, Accuracy: 0.8097560975609757
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 4, Activation: Softmax, Accuracy: 0.83414634146341
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 8, Activation: ReLU, Accuracy: 0.8926829268292683
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 8, Activation: Sigmoid, Accuracy: 0.80975609756097
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 8, Activation: Tanh, Accuracy: 0.8390243902439024
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 8, Activation: Softmax, Accuracy: 0.87804878048780
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 16, Activation: ReLU, Accuracy: 0.9853658536585366
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 16, Activation: Sigmoid, Accuracy: 0.7853658536585
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 16, Activation: Tanh, Accuracy: 0.9073170731707317
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 16, Activation: Softmax, Accuracy: 0.8878048780487
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 32, Activation: ReLU, Accuracy: 0.9853658536585366
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 32, Activation: Sigmoid, Accuracy: 0.7853658536585
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 32, Activation: Tanh, Accuracy: 0.975609756097561
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 32, Activation: Softmax, Accuracy: 0.9073170731707
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 64, Activation: ReLU, Accuracy: 0.9853658536585366
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 64, Activation: Sigmoid, Accuracy: 0.8048780487804
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 64, Activation: Tanh, Accuracy: 0.9853658536585366
Epoch: 50, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 64, Activation: Softmax, Accuracy: 0.9268292682926
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 4, Activation: ReLU, Accuracy: 0.8341463414634146
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 4, Activation: Sigmoid, Accuracy: 0.785365853658536
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 4, Activation: Tanh, Accuracy: 0.7951219512195122
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 4, Activation: Softmax, Accuracy: 0.8195121951219512
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 8, Activation: ReLU, Accuracy: 0.8146341463414634
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 8, Activation: Sigmoid, Accuracy: 0.8
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 8, Activation: Tanh, Accuracy: 0.8195121951219512
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 8, Activation: Softmax, Accuracy: 0.8195121951219512
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 16, Activation: ReLU, Accuracy: 0.8585365853658536
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 16, Activation: Sigmoid, Accuracy: 0.7951219512195
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 16, Activation: Tanh, Accuracy: 0.8195121951219512
Epoch: 50, LR: 0.001, Batch Size: 32, Hidden Layers: 1, Units: 16, Activation: Softmax, Accuracy: 0.8292682926829
```

Penjelasan:

Kode yang Anda berikan merupakan sebuah **loop bersarang** yang dirancang untuk **menguji berbagai konfigurasi** dari sebuah model jaringan saraf tiruan (JSTT) tipe Multilayer Perceptron (MLP). Tujuannya adalah untuk menemukan konfigurasi yang menghasilkan akurasi terbaik pada data uji.
Contoh output Epoch: 59, LR: 0.001, Batch Size: 16, Hidden Layers: 3, Units: 64, Activation: Tanh, Accuracy: 0.9853658536585366 :
Artinya, pada iterasi ke-59, dengan learning rate 0.001, batch size 16, 3 hidden layer, 64 unit pada hidden layer terakhir, dan fungsi aktivasi Tanh, model mencapai akurasi 98.54% pada data uji.

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

Output:

```
Best Model Accuracy: 1.0
```

Penjelasan:
Kode ini memberikan cara yang sederhana untuk menampilkan hasil evaluasi akhir dari sebuah model machine learning, khususnya nilai akurasi terbaik. Namun, untuk mendapatkan pemahaman yang lebih komprehensif tentang kinerja model, perlu dilakukan analisis lebih lanjut dengan mempertimbangkan berbagai faktor seperti ukuran dataset, kompleksitas model, dan metrik evaluasi lainnya.

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

Penjelasan:
Kode torch.save(best_model.state_dict(), 'best_heart_model.pth') digunakan dalam PyTorch untuk menyimpan model yang telah dilatih ke dalam file.