Nama : Az – Zahra Chikal E

NIM : 1103213039

Kelas : TK-45-05

TUGAS WEEK 11 MLP DEEP LEARNING DUMMY DATA

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

✓ 4.0s Python

```python
# Generate dummy data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)
```

✓ 0.0s Python

```python
# Define the MLP model
class MLPModel(nn.Module):
    def __init__(self, input_dim, hidden_layers, activation_function):
        super(MLPModel, self).__init__()
        layers = []
        current_dim = input_dim

        # Add hidden layers
        for hidden_neurons in hidden_layers:
            layers.append(nn.Linear(current_dim, hidden_neurons))
            if activation_function == 'relu':
                layers.append(nn.ReLU())
            elif activation_function == 'sigmoid':
                layers.append(nn.Sigmoid())
            elif activation_function == 'tanh':
                layers.append(nn.Tanh())
            elif activation_function == 'linear':
                pass  # No activation for linear
            elif activation_function == 'softmax':
                layers.append(nn.Softmax(dim=1))
            current_dim = hidden_neurons

        # Add output layer
        layers.append(nn.Linear(current_dim, 1))
        layers.append(nn.Sigmoid())  # Binary classification
        self.network = nn.Sequential(*layers)

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

✓ 0.0s Python

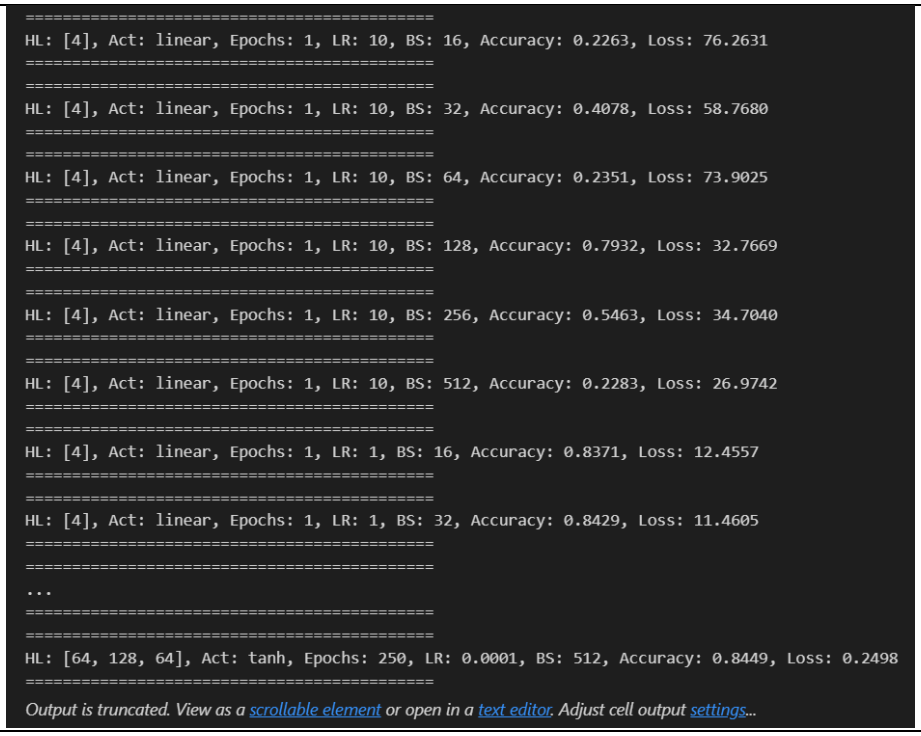
Analisis:

Pada kode diatas digunakan untuk membuat model neural network sederhana untuk tugas klasifikasi biner. Data dibuat secara dummy (buatan) menggunakan fungsi make_classification dari library sklearn. Dataset ini berisi 1000 sampel dengan 20 fitur untuk dua kelas. Data kemudian dinormalisasi menggunakan StandardScaler agar lebih mudah diproses oleh model. Setelah itu, data dibagi menjadi data latih (80%) dan data uji (20%) menggunakan fungsi train_test_split. Data ini selanjutnya diubah menjadi bentuk tensor PyTorch agar bisa digunakan dalam proses pelatihan model. Model neural network yang

dibuat disebut Multi-Layer Perceptron (MLP). Model ini didefinisikan menggunakan kelas khusus bernama MLPModel. Model ini dirancang fleksibel, sehingga pengguna bisa menentukan jumlah layer tersembunyi (hidden layers), jumlah neuron di setiap layer, dan fungsi aktivasi seperti ReLU, sigmoid, atau tanh. Fungsi aktivasi membantu model belajar pola yang lebih kompleks dari data. Bagian akhir model menggunakan fungsi aktivasi sigmoid untuk mengeluarkan nilai probabilitas, karena tugas yang dilakukan adalah memutuskan apakah data termasuk kelas 0 atau 1.

```python
# Hyperparameter grid
hidden_layers_options = [
    [4], [8], [16], [32], [64],    # 1 hidden layer
    [4, 4], [8, 8], [16, 16], [32, 32], [64, 64],  # 2 hidden layers
    [4, 8, 4], [8, 16, 8], [16, 32, 16], [32, 64, 32], [64, 128, 64] # 3 hidden layers
]
activation_options = ['linear', 'sigmoid', 'relu', 'softmax', 'tanh']
epoch_options = [1, 10, 25, 50, 100, 250]
learning_rate_options = [10, 1, 0.1, 0.01, 0.001, 0.0001]
batch_size_options = [16, 32, 64, 128, 256, 512]

# Save configuration for progress tracking
results = []

# Function to evaluate model
def train_and_evaluate(hidden_layers, activation_function, epochs, learning_rate, batch_size):
    # Create DataLoader for training data
    train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

    # Initialize model, loss, and optimizer
    model = MLPModel(input_dim=X_train_tensor.shape[1], hidden_layers=hidden_layers, activation_function=activation_function)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Training loop
    model.train()
    epoch_losses = []
    for epoch in range(epochs):
        total_loss = 0
        for batch_X, batch_y in train_loader:
            optimizer.zero_grad()
            predictions = model(batch_X)
            loss = criterion(predictions, batch_y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        # Calculate average loss for this epoch
        avg_loss = total_loss / len(train_loader)
        epoch_losses.append(avg_loss)

    # Evaluation
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_X, batch_y in train_loader:  # Using the entire dataset for evaluation
            predictions = model(batch_X)
            predicted = (predictions > 0.5).float()
            correct += (predicted == batch_y).sum().item()
            total += batch_y.size(0)
    accuracy = correct / total
    return accuracy, epoch_losses

# Running the grid search for hyperparameters
for hidden_layers in hidden_layers_options:
    for activation in activation_options:
        for epochs in epoch_options:
            for lr in learning_rate_options:
                for batch_size in batch_size_options:
                    # Train and evaluate the model
                    accuracy, losses = train_and_evaluate(
                        hidden_layers=hidden_layers,
                        activation_function=activation,
                        epochs=epochs,
                        learning_rate=lr,
                        batch_size=batch_size
                    )

                    # Store results
                    results.append({
                        'hidden_layers': hidden_layers,
                        'activation': activation,
                        'epochs': epochs,
                        'learning_rate': lr,
                        'batch_size': batch_size,
                        'accuracy': accuracy,
                        'loss': losses
                    })
                    print("=======================================")
                    print(f"HL: {hidden_layers}, Act: {activation}, Epochs: {epochs}, LR: {lr}, BS: {batch_size}, Accuracy: {accuracy:.4f}, Loss: {losses[-1]:.4f}")
                    print("=======================================")

# Save the results to a CSV file
data = pd.DataFrame(results)
data.to_csv("mlp_experiment_Deep_Learning_Heart_results.csv", mode='w', header=True, index=False)
```

Output:

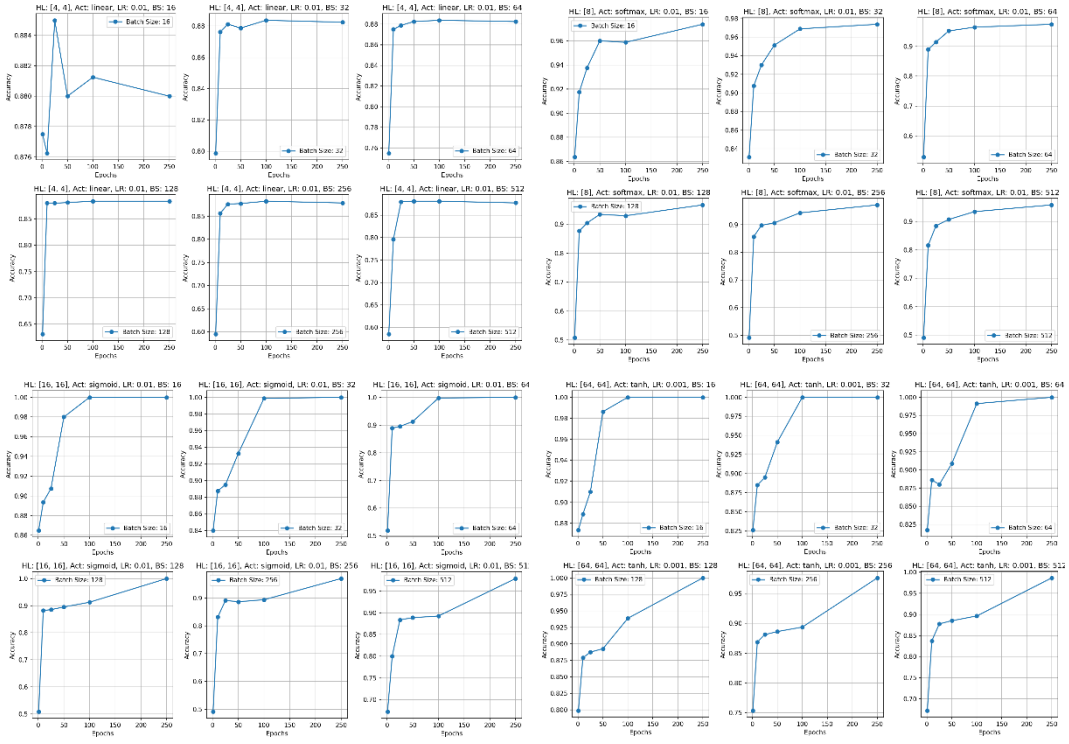


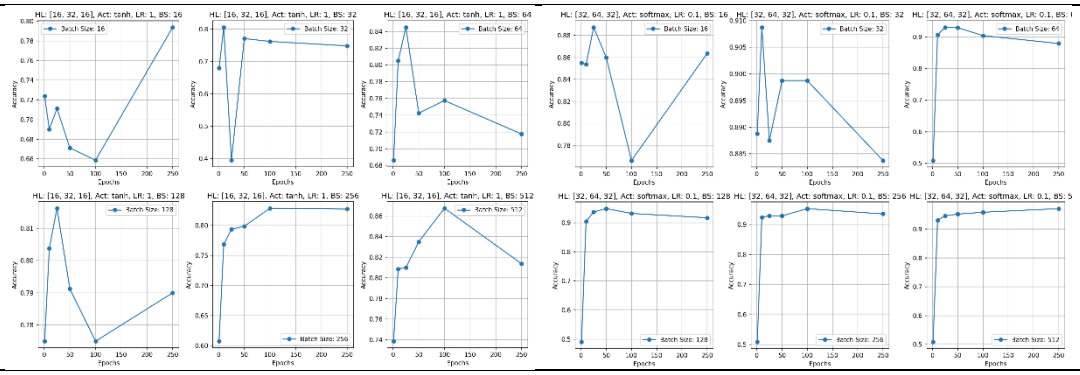
Analisis:

Pada kode diatas digunakan untuk melakukan experiment dengan beberapa parameter. hidden_layers_options digunakan untuk menentukan jumlah dan ukuran lapisan tersembunyi yang berbeda, mulai dari satu hingga tiga lapisan dengan variasi jumlah neuron. activation_options berisi berbagai fungsi aktivasi yang dapat digunakan, seperti ReLU, Sigmoid, Tanh, Softmax, dan Linear. epoch_options digunakan untuk menentukan jumlah epoch (iterasi pelatihan) yang berbeda untuk model. learning_rate_options berisi berbagai nilai learning rate yang mempengaruhi seberapa cepat model belajar. batch_size_options digunakan untuk menentukan ukuran batch untuk pelatihan. Proses ini untuk mengeksplorasi berbagai konfigurasi model secara sistematis dan menemukan pengaturan optimal untuk meningkatkan performa klasifikasi penyakit jantung. Hasilnya akan disimpan di file cdv yaitu ’mlp_experiment_Deep_Learning_Heart_results.csv’.

```python
import matplotlib.pyplot as plt
import os

# Load hasil eksperimen dari file CSV
data = pd.read_csv("mlp_experiment_Deep_Learning_Heart_results.csv")

# Buat folder untuk menyimpan hasil plot jika belum ada
output_folder = "mlp_experiment_Deep Learning(Heart)_plots"
os.makedirs(output_folder, exist_ok=True)

# Hyperparameter grid untuk visualisasi
hidden_layers_list = [
    str([4]), str([8]), str([16]), str([32]), str([64]),  # 1 hidden layer
    str([4, 4]), str([8, 8]), str([16, 16]), str([32, 32]), str([64, 64]),  # 2 hidden layers
    str([4, 8, 4]), str([8, 16, 8]), str([16, 32, 16]), str([32, 64, 32]),  # 3 hidden layers
]
activations_list = ['linear', 'sigmoid', 'relu', 'softmax', 'tanh']
learning_rates = [10, 1, 0.1, 0.01, 0.001, 0.0001]

# Loop melalui kombinasi parameter
for hidden_layers in hidden_layers_list:
    for activation in activations_list:
        for lr in learning_rates:
            # Filter data untuk kombinasi parameter ini
            filtered_data = data[
                (data['hidden_layers'] == hidden_layers) &
                (data['activation'] == activation) &
                (data['learning_rate'] == lr)
            ]

            if not filtered_data.empty:
                # Plot akurasi terhadap epochs untuk setiap batch size
                batch_sizes = filtered_data['batch_size'].unique()

                plt.figure(figsize=(12, 12))
                for i, batch_size in enumerate(batch_sizes, 1):
                    plt.subplot(3, 3, i)  # Buat grid 3x3
                    batch_data = filtered_data[filtered_data['batch_size'] == batch_size]
                    plt.plot(batch_data['epochs'], batch_data['accuracy'], marker='o', label=f'Batch Size: {batch_size}')
                    plt.title(f'HL: {hidden_layers}, Act: {activation}, LR: {lr}, BS: {batch_size}')
                    plt.xlabel('Epochs')
                    plt.ylabel('Accuracy')
                    plt.legend()
                    plt.grid()

                plt.tight_layout()

                # Nama file berdasarkan kombinasi hyperparameter
                filename = f"HL-{hidden_layers}_Act-{activation}_LR-{lr}.png"
                filepath = os.path.join(output_folder, filename)

                # Simpan plot ke file
                plt.savefig(filepath, dpi=150)
                plt.close()
```

Output:





Analisis:

Pada kode diatas digunakan untuk memuat hasil eksperimen dari file CSV, kemudian menghasilkan dan menyimpan grafik yang menunjukkan akurasi model terhadap jumlah epoch untuk berbagai kombinasi hyperparameter (jumlah hidden layers, fungsi aktivasi, dan learning rate) serta ukuran batch. Kode ini menggunakan Pandas untuk manipulasi data dan Matplotlib untuk visualisasi. Pertama, data dibaca dari file CSV, dan folder output dibuat jika belum ada. Selanjutnya, kode melakukan loop melalui semua kombinasi hyperparameter yang ditentukan, memfilter data berdasarkan kombinasi tersebut, dan jika data tidak kosong, menghasilkan subplot untuk setiap ukuran batch yang unik. Setiap grafik disimpan dengan nama file yang mencerminkan pengaturan hyperparameter. Meskipun grafik dihasilkan untuk kombinasi yang berbeda, beberapa grafik dapat terlihat sama karena dataset yang digunakan memiliki karakteristik serupa atau jika model dilatih pada subset data yang identik, hasil akurasi bisa serupa. kemudian, interaksi antara hyperparameter tertentu dapat menyebabkan model berperilaku mirip meskipun ada perbedaan dalam pengaturannya. Lalu jika model mengalami overfitting atau jika variasi dalam akurasi selama pelatihan tidak cukup besar untuk dibedakan secara visual.

```python
import pandas as pd

# Load the experiment results from the CSV file
results_df = pd.read_csv("mlp_experiment_Deep_Learning_Heart_results.csv")

# Sort the results by accuracy in descending order
top_results = results_df.sort_values(by="accuracy", ascending=False).head(10)

# Display the top 10 results
top_results
```

Output:

	hidden_layers	activation	epochs	learning_rate	batch_size	accuracy	loss
9590	[32, 32]	tanh	25	0.10	64	1.0	[0.4598642649019466, 0.37076738126137676, 0.25...
14994	[32, 64, 32]	tanh	25	0.01	16	1.0	[0.43525133293408613, 0.36517155158978243, 0.3...
9232	[32, 32]	relu	100	0.10	256	1.0	[0.4337046753615141, 0.28677146836416795, 0.28...
10131	[64, 64]	sigmoid	250	0.10	128	1.0	[0.8864160312546624, 0.5955139431688521, 0.494...
15606	[64, 128, 64]	relu	10	0.01	16	1.0	[0.37813295733538743, 0.2841247306420253, 0.20...
14564	[32, 64, 32]	relu	25	0.01	64	1.0	[0.4415639764903223, 0.31688060523832545, 0.26...
13557	[16, 32, 16]	relu	100	0.01	128	1.0	[0.6540074282222323, 0.4421088877651427, 0.389...
13556	[16, 32, 16]	relu	100	0.01	64	1.0	[0.5754646641366622, 0.4582844621994916, 0.344...
13555	[16, 32, 16]	relu	100	0.01	32	1.0	[0.5176854770291935, 0.35334398236238596, 0.31...
13554	[16, 32, 16]	relu	100	0.01	16	1.0	[0.4583285092161251 7, 0.2988141804933548, 0.27...

Analisis:

Pada kode diatas digunakan untuk menampilkan 10 nilai yang terbagus/terbaik dari hiden layer 1, hiden layer 2, hiden layer 3. dapat diketahui bahwa nilai 10 terbaik berada di hiden layer 3 dan hiden layer 2 dengan akurasi sebesar 1.