Nama: Andi Cleopatra Maryam jamila

Nim: 1103213071

Week 11 Classification dummy data

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

```
# Membuat dummy data untuk klasifikasi
# Fitur X berjumlah 1000 data, 20 fitur
X = np.random.rand(1000, 20)

# Membuat target y yang terdiri dari dua kelas (0 atau 1)
y = np.random.randint(0, 2, size=(1000,))

# Membagi 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)

# Normalisasi fitur menggunakan StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Konversi data ke format 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)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

```
# Definisi model MLP
class MLPModel(nn.Module):
    def __init__(self, input_dim, hidden_layers, hidden_units, activation_function):
        super(MLPModel, self).__init__()

        # Menyusun layers berdasarkan parameter eksperimen
        layers = []
        prev_units = input_dim
        for _ in range(hidden_layers):
            layers.append(nn.Linear(prev_units, hidden_units))  # Menambahkan layer Linear
            # Menambahkan fungsi aktivasi sesuai parameter
            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 == 'Softmax':
                layers.append(nn.Softmax(dim=1))
            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)
```

```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):
    dataset = TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

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

    criterion = nn.CrossEntropyLoss()  # CrossEntropyLoss untuk klasifikasi
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    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
```

Analisis:

Melakukan pembuatan model klasifikasi menggunakan deep learning dengan arsitektur Multi-Layer Perceptron (MLP) menggunakan PyTorch. Data dummy dengan 1000 sampel dan 20 fitur dihasilkan secara acak, dan kemudian dibagi menjadi data pelatihan dan pengujian (80% dan 20%, masing-masing). Data fitur dinormalisasi menggunakan StandardScaler untuk memastikan bahwa input model berada dalam rentang yang optimal. Model MLP dibangun dengan jumlah layer tersembunyi dan jumlah neuron yang dapat disesuaikan, serta fungsi aktivasi yang bisa dipilih dari ReLU, Sigmoid, Tanh, atau Softmax. Proses pelatihan dilakukan dengan menggunakan CrossEntropyLoss sebagai fungsi loss dan Adam sebagai optimizer untuk mengupdate bobot model. Proses pelatihan mencakup beberapa epoch dan ukuran batch yang dapat disesuaikan. Model yang dilatih ini akan dievaluasi berdasarkan akurasi yang dihitung pada data pengujian. Kode ini menyediakan fleksibilitas untuk menguji berbagai kombinasi hyperparameter, seperti jumlah hidden layer, jumlah neuron, fungsi aktivasi, learning rate, dan batch size untuk menemukan konfigurasi terbaik.

```python
# Parameter eksperimen yang akan diuji
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']

best_model = None
best_accuracy = 0

# Melatih dan menguji model dengan berbagai kombinasi parameter
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
                        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: 10, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: ReLU, Accuracy: 0.62
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Sigmoid, Accuracy: 0.525
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Tanh, Accuracy: 0.545
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 4, Activation: Softmax, Accuracy: 0.58
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: ReLU, Accuracy: 0.39
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Sigmoid, Accuracy: 0.595
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Tanh, Accuracy: 0.56
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 8, Activation: Softmax, Accuracy: 0.54
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: ReLU, Accuracy: 0.5
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Sigmoid, Accuracy: 0.52
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Tanh, Accuracy: 0.52
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 16, Activation: Softmax, Accuracy: 0.595
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: ReLU, Accuracy: 0.525
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Sigmoid, Accuracy: 0.58
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Tanh, Accuracy: 0.405
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 32, Activation: Softmax, Accuracy: 0.48
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: ReLU, Accuracy: 0.5
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Sigmoid, Accuracy: 0.405
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Tanh, Accuracy: 0.475
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 1, Units: 64, Activation: Softmax, Accuracy: 0.5
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 4, Activation: ReLU, Accuracy: 0.37
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 4, Activation: Sigmoid, Accuracy: 0.37
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 4, Activation: Tanh, Accuracy: 0.59
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 4, Activation: Softmax, Accuracy: 0.63
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 8, Activation: ReLU, Accuracy: 0.37
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 8, Activation: Sigmoid, Accuracy: 0.63
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 8, Activation: Tanh, Accuracy: 0.54
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 8, Activation: Softmax, Accuracy: 0.63
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 16, Activation: ReLU, Accuracy: 0.37
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 16, Activation: Sigmoid, Accuracy: 0.37
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 16, Activation: Tanh, Accuracy: 0.505
Epoch: 1, LR: 10, Batch Size: 16, Hidden Layers: 2, Units: 16, Activation: Softmax, Accuracy: 0.385
```

Analisis:

Melakukan pencarian hyperparameter terbaik dengan menguji berbagai kombinasi parameter dalam pelatihan model klasifikasi menggunakan MLP di PyTorch. Eksperimen ini menguji kombinasi jumlah epoch, learning rate, batch size, jumlah hidden layer, jumlah neuron di setiap layer, dan fungsi aktivasi (ReLU, Sigmoid, Tanh, Softmax) untuk menemukan model dengan akurasi tertinggi. Setiap kombinasi parameter diuji dengan melatih model menggunakan fungsi train_model, diikuti dengan evaluasi akurasi pada data uji. Akurasi model dihitung dengan membandingkan prediksi terhadap label yang benar. Model dengan akurasi terbaik disimpan untuk digunakan lebih lanjut. Dengan cara ini, proses tuning dilakukan secara menyeluruh untuk menentukan konfigurasi yang optimal bagi model, dan hasilnya dapat membantu memilih model yang paling efektif untuk tugas klasifikasi tersebut.

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

Analisis:
Menyimpan model terbaik yang telah dilatih dalam file dengan format .pth menggunakan fungsi torch.save(). best_model.state_dict() menyimpan bobot dan parameter dari model terbaik yang ditemukan selama eksperimen, termasuk konfigurasi lapisan dan bobot yang diperoleh setelah proses pelatihan. Menyimpan model dengan cara ini memungkinkan model untuk dimuat kembali di masa depan tanpa perlu dilatih ulang, sehingga menghemat waktu dan sumber daya. File best_classification_model.pth dapat digunakan untuk evaluasi lebih lanjut atau untuk penerapan model dalam aplikasi dunia nyata, memastikan bahwa hasil eksperimen dapat dipertahankan dan diterapkan dengan efisien.