Laporan Dummy Data

Nama : Anita Firda Nuralifah

NIM : 1103213117

Library

```
import pandas as pd #untuk manipulasi dan analisis data.
import numpy as np #untuk komputasi numerik dengan array dan matriks.
import torch #untuk membangun dan melatih model neural network.
import torch.nn as nn #Modul PyTorch yang berisi berbagai lapisan dan fungsi untuk membangun neural network.
import torch.optim as optim #Modul PyTorch yang berisi berbagai algoritma optimasi untuk melatih model neural network.
from torch.utils.data import DataLoader, TensorDataset #untuk membuat dan mengelola dataset, serta memuat data secara efisien selama pelatihan.
from sklearn.datasets import make_classification #untuk membuat dataset sintetis, seperti dataset klasifikasi.
from sklearn.model_selection import train_test_split #untuk membagi dataset menjadi data latih dan data uji.
from sklearn.preprocessing import StandardScaler #untuk melakukan preprocessing data, seperti standarisasi fitur.
from sklearn.metrics import accuracy_score #untuk menghitung akurasi model klasifikasi.
```

- import pandas as pd : untuk manipulasi dan analisis data.
- import numpy as np : untuk komputasi numerik dengan array dan matriks.
- import torch : untuk membangun dan melatih model neural network.
- import torch.nn as nn : Modul PyTorch yang berisi berbagai lapisan dan fungsi untuk membangun neural network.
- import torch.optim as optim : Modul PyTorch yang berisi berbagai algoritma optimasi untuk melatih model neural network.
- from torch.utils.data import DataLoader, TensorDataset : untuk membuat dan mengelola dataset, serta memuat data secara efisien selama pelatihan.
- from sklearn.datasets import make_classification : untuk membuat dataset sintetis, seperti dataset klasifikasi.
- from sklearn.model_selection import train_test_split : untuk membagi dataset menjadi data latih dan data uji.
- from sklearn.preprocessing import StandardScaler : untuk melakukan preprocessing data, seperti standarisasi fitur.
- from sklearn.metrics import accuracy_score : untuk menghitung akurasi model klasifikasi.

```
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)
#membuat data dummy untuk klasifikasi data

scaler = StandardScaler()
X = scaler.fit_transform(X)
#pemrosesan data
```

Kode ini membuat dataset simulasi untuk klasifikasi menggunakan make_classification, dengan 1000 sampel, 20 fitur, dan 2 kelas. Fitur X kemudian dinormalisasi menggunakan StandardScaler agar memiliki rata-rata 0 dan standar deviasi 1, untuk meningkatkan kinerja algoritma machine learning.

```
[3] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    #membagi data menjadi data latih dan data uji
```

Kode ini menggunakan fungsi train_test_split dari sklearn untuk membagi dataset menjadi data latih (training) dan data uji (testing). Parameter test_size=0.3 menunjukkan bahwa 30% data digunakan sebagai data uji, sementara 70% sisanya sebagai data latih. Parameter

random_state=42 memastikan pembagian data bersifat konsisten setiap kali kode dijalankan. Hasil pembagian disimpan dalam variabel X_train, X_test (fitur) dan y_train, y_test (label).

```
[4] X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train, dtype=torch.long)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_test_tensor = torch.tensor(y_test, dtype=torch.long)
    #mengonversi data ke tensors PyTorch
```

Kode ini mengonversi data latih (X_train, y_train) dan data uji (X_test, y_test) ke dalam bentuk tensor menggunakan PyTorch. Fitur (X_train dan X_test) dikonversi ke tipe float32, sedangkan label (y_train dan y_test) dikonversi ke tipe long. Proses ini diperlukan agar data dapat digunakan dalam model berbasis PyTorch.

```
def create_mlp(input_size, hidden_layers, hidden_neurons, activation_function):
    layers = []
    layers.append(nn.Linear(input_size, hidden_neurons))
    for _ in range(hidden_layers - 1):
        layers.append(nn.Linear(hidden_neurons, hidden_neurons))
    #menambahkan lapisan tersembunyi

    if activation_function == 'linear':
        activation = nn.Identity()
    elif activation_function == 'Sigmoid':
        activation = nn.Sigmoid()
    elif activation_function == 'ReLU':
        activation = nn.ReLU()
    elif activation_function == 'Softmax':
        activation = nn.Softmax(dim=1)
    elif activation_function == 'Tanh':
        activation = nn.Tanh()
    #memilih fungsi aktivasi

    layers.append(nn.Linear(hidden_neurons, 2))

    model = nn.Sequential(*layers)
    return model
    #mnggabungkan semua lapisan


#menentukan model MLP
```

Menentukan model MLP

```python
def train_and_evaluate(model, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, batch_size, epochs, learning_rate):
    train_data = TensorDataset(X_train_tensor, y_train_tensor)
    train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)

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

    model.train()
    for epoch in range(epochs):
        for X_batch, y_batch in train_loader:
            optimizer.zero_grad()
            output = model(X_batch)
            loss = criterion(output, y_batch)
            loss.backward()
            optimizer.step()
    #mengatur model ke mode pelatihan

    model.eval()
    with torch.no_grad():
        train_output = model(X_train_tensor)
        train_pred = torch.argmax(train_output, dim=1)
        train_accuracy = accuracy_score(y_train_tensor.numpy(), train_pred.numpy())
        train_loss = criterion(train_output, y_train_tensor).item()
        #akurasi pelatihan

        test_output = model(X_test_tensor)
        test_pred = torch.argmax(test_output, dim=1)
        test_accuracy = accuracy_score(y_test_tensor.numpy(), test_pred.numpy())
        test_loss = criterion(test_output, y_test_tensor).item()
        #akurasi pengujian
        #mengatur model ke mode evaluasi

    return train_accuracy, train_loss, test_accuracy, test_loss
     #mengembalikan hasil

#melatih dan mengevaluasi model
```

Melatih dan mengevaluasi model

```python
hidden_layers_options = [1, 2, 3]
hidden_neurons_options = [4, 8, 16, 32, 64]
activation_functions = ['linear', 'Sigmoid', 'ReLU', 'Softmax', 'Tanh']
epochs_options = [25, 50, 100, 250]
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [64, 128, 256, 512]
#eksperimen menggunakan hyperparameter

results = []
best_result = None
worst_result = None
best_per_activation = {activation: None for activation in activation_functions}
#melacak hasil eksperimen

for hidden_layers in hidden_layers_options:
    for hidden_neurons in hidden_neurons_options:
        for activation_function in activation_functions:
            for epochs in epochs_options:
                for learning_rate in learning_rates:
                    for batch_size in batch_sizes:
                        print(f"Melatih dengan {hidden_layers} lapisan tersembunyi, {hidden_neurons} neuron, fungsi aktivasi {activation_function}, "
                              f"{epochs} epoch, tingkat pembelajaran {learning_rate}, ukuran batch {batch_size}")
                        #mencetak konfigurasi eksperimen saat ini

                        model = create_mlp(X_train_tensor.shape[1], hidden_layers, hidden_neurons, activation_function)
                        #membuat model untuk setiap kombinasi hyperparameter

                        train_acc, train_loss, test_acc, test_loss = train_and_evaluate(
                            model, X_train_tensor, y_train_tensor, X_test_tensor, y_test_tensor, batch_size, epochs, learning_rate
                        )
                        #melatih dan mengevaluasi model

                        results.append({
                            'Hidden Layers': hidden_layers,
                            'Hidden Neurons': hidden_neurons,
                            'Activation Function': activation_function,
                            'Epochs': epochs,
                            'Learning Rate': learning_rate,
                            'Batch Size': batch_size,
                            'Train Accuracy': train_acc,
                            'Train Loss': train_loss,
                            'Test Accuracy': test_acc,
                            'Test Loss': test_loss
                        })
                        #menyimpan hasil

                        print(f"Akurasi Pelatihan: {train_acc * 100:.2f}%")
                        print(f"Loss Pelatihan: {train_loss:.4f}")
                        print(f"Akurasi Pengujian: {test_acc * 100:.2f}%")
                        print(f"Loss Pengujian: {test_loss:.4f}\n")
                        #mencetak hasil akhir dari eksperimen ini


                        if best_result is None or test_acc > best_result['Test Accuracy']:
                            best_result = {
                                'Hyperparameters': {
                                    'Hidden Layers': hidden_layers,
                                    'Hidden Neurons': hidden_neurons,
                                    'Activation Function': activation_function,
                                    'Epochs': epochs,
                                    'Learning Rate': learning_rate,
                                    'Batch Size': batch_size
                                },
                                'Test Accuracy': test_acc
                            }
                        if worst_result is None or test_acc < worst_result['Test Accuracy']:
                            worst_result = {
                                'Hyperparameters': {
                                    'Hidden Layers': hidden_layers,
                                    'Hidden Neurons': hidden_neurons,
                                    'Activation Function': activation_function,
                                    'Epochs': epochs,
                                    'Learning Rate': learning_rate,
                                    'Batch Size': batch_size
                                },
                                'Test Accuracy': test_acc
                            }
                         #memperbarui hasil terbaik dan terburuk

                        if best_per_activation[activation_function] is None or test_acc > best_per_activation[activation_function]['Test Accuracy']:
                            best_per_activation[activation_function] = {
                                'Hyperparameters': {
                                    'Hidden Layers': hidden_layers,
                                    'Hidden Neurons': hidden_neurons,
                                    'Activation Function': activation_function,
                                    'Epochs': epochs,
                                    'Learning Rate': learning_rate,
                                    'Batch Size': batch_size
                                },
                                'Test Accuracy': test_acc
                            }
                          #melacak konfigurasi terbaik untuk setiap fungsi aktivasi

#loop melalui semua kombinasi hyperparameter
```

Melakukan pengujian parameter

```
[10] print("\nBest Hyperparameter Configuration (Overall):")
     print(f"Hidden Layers: {best_result['Hyperparameters']['Hidden Layers']}")
     print(f"Hidden Neurons: {best_result['Hyperparameters']['Hidden Neurons']}")
     print(f"Activation Function: {best_result['Hyperparameters']['Activation Function']}")
     print(f"Epochs: {best_result['Hyperparameters']['Epochs']}")
     print(f"Learning Rate: {best_result['Hyperparameters']['Learning Rate']}")
     print(f"Batch Size: {best_result['Hyperparameters']['Batch Size']}")
     print(f"Test Accuracy: {best_result['Test Accuracy'] * 100:.2f}%")
     #menampilkan Konfigurasi Hyperparameter Terbaik
```

```
Best Hyperparameter Configuration (Overall):
Hidden Layers: 1
Hidden Neurons: 4
Activation Function: Softmax
Epochs: 25
Learning Rate: 0.1
Batch Size: 128
Test Accuracy: 87.00%
```

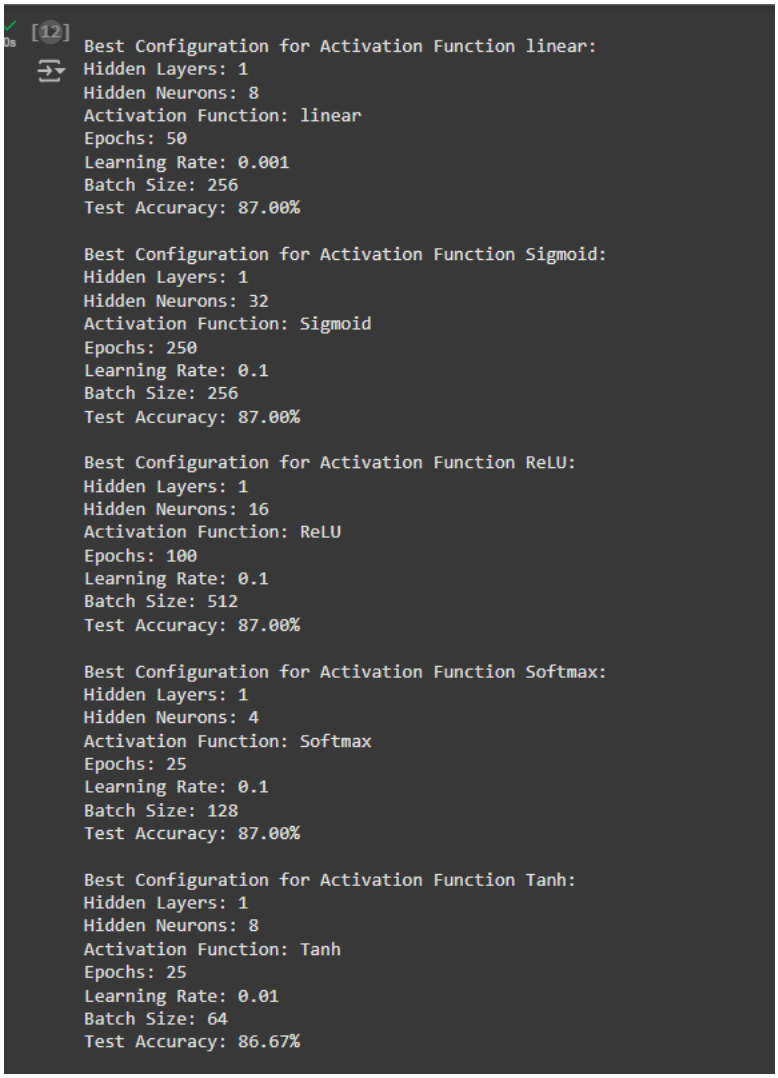
Menampilkan parameter terbaik

```
print("\nWorst Hyperparameter Configuration:")
print(f"Hidden Layers: {worst_result['Hyperparameters']['Hidden Layers']}")
print(f"Hidden Neurons: {worst_result['Hyperparameters']['Hidden Neurons']}")
print(f"Activation Function: {worst_result['Hyperparameters']['Activation Function']}")
print(f"Epochs: {worst_result['Hyperparameters']['Epochs']}")
print(f"Learning Rate: {worst_result['Hyperparameters']['Learning Rate']}")
print(f"Batch Size: {worst_result['Hyperparameters']['Batch Size']}")
print(f"Test Accuracy: {worst_result['Test Accuracy'] * 100:.2f}%")
#menampilkan Konfigurasi Hyperparameter Terburuk
```

```
Worst Hyperparameter Configuration:
Hidden Layers: 3
Hidden Neurons: 64
Activation Function: linear
Epochs: 25
Learning Rate: 0.1
Batch Size: 512
Test Accuracy: 23.67%
```

Menampilkan parameter teburuk

```
for activation in activation_functions:
    print(f"\nBest Configuration for Activation Function {activation}:")
    if best_per_activation[activation]:
        print(f"Hidden Layers: {best_per_activation[activation]['Hyperparameters']['Hidden Layers']}")
        print(f"Hidden Neurons: {best_per_activation[activation]['Hyperparameters']['Hidden Neurons']}")
        print(f"Activation Function: {activation}")
        print(f"Epochs: {best_per_activation[activation]['Hyperparameters']['Epochs']}")
        print(f"Learning Rate: {best_per_activation[activation]['Hyperparameters']['Learning Rate']}")
        print(f"Batch Size: {best_per_activation[activation]['Hyperparameters']['Batch Size']}")
        print(f"Test Accuracy: {best_per_activation[activation]['Test Accuracy'] * 100:.2f}%")
    else:
        print("No result for this activation function.")
#menampilkan Konfigurasi Terbaik untuk Setiap Fungsi Aktivasi
```



Menampilkan Konfigurasi Terbaik untuk Setiap Fungsi Aktivasi