Rizqy Asyraff Athallah

110321058

Week 11 Deep Learning

- **Prepocessing**

```
# Memisahkan fitur (X) dan target (y)
X = data.drop(columns=['target'])  # Menghapus kolom target dari dataset
y = data['target']                 # Menyimpan kolom target ke variabel y

# Normalisasi fitur menggunakan StandardScaler
scaler = StandardScaler()          # Inisialisasi scaler
X_scaled = scaler.fit_transform(X) # Menormalisasi semua fitur

# Membagi dataset menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

- **Memisahkan Fitur dan Target**: Kolom target adalah variabel target (klasifikasi), sementara kolom lainnya adalah fitur.
- **Normalisasi**: Data fitur dinormalisasi menggunakan StandardScaler untuk meningkatkan performa pelatihan model.
- **Split Dataset**: Dataset dibagi menjadi data *training* dan *testing* dengan rasio 80:20 untuk melatih dan mengevaluasi model.

- **Konversi ke PyTorch Tensors**

```
# Mengonversi data ke dalam bentuk tensor PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)  # Tensor fitur training
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)    # Tensor fitur testing
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)  # Tensor target training
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)    # Tensor target testing
```

- Dataset diubah menjadi format tensor PyTorch untuk digunakan dalam pelatihan model.
- **dtype=torch.float32**: Semua fitur disimpan dalam format *floating point*.
- **dtype=torch.long**: Target dikonversi menjadi bilangan bulat karena fungsi *loss* membutuhkan tipe ini untuk klasifikasi.

- **Membuat Custom Dataset**

```
# Mendefinisikan dataset custom
class HeartDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features  # Menyimpan fitur
        self.labels = labels      # Menyimpan target

    def __len__(self):
```

```
        return len(self.features)  # Mengembalikan panjang dataset

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]  # Mengembalikan item berdasarkan indeks

# Membuat objek dataset untuk training dan testing
dataset_train = HeartDataset(X_train_tensor, y_train_tensor)
dataset_test = HeartDataset(X_test_tensor, y_test_tensor)
```

- **Custom Dataset**: Digunakan untuk mendefinisikan bagaimana data diakses dalam batch saat pelatihan.
- **Metode** : __len__: Mengembalikan jumlah sampel dalam dataset, dan __getitem__: Mengembalikan fitur dan label pada indeks tertentu.
- Dataset untuk *training* dan *testing* dibuat menggunakan class HeartDataset.

➢ Membangun MLP Model

```
# Mendefinisikan MLP model
class MLPClassifier(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(MLPClassifier, self).__init__()
        layers = []
        in_features = input_size

        # Membuat hidden layers
        for hidden_units in hidden_layers:
            layers.append(nn.Linear(in_features, hidden_units))  # Linear Layer
            layers.append(activation_fn)  # Activation Function
            in_features = hidden_units

        # Output layer
        layers.append(nn.Linear(in_features, 2))  # Binary classification (2 classes)
        self.model = nn.Sequential(*layers)  # Menggabungkan semua layer ke dalam Sequential

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

- **MLPClassifier**: Model berbasis Multi-Layer Perceptron (MLP) untuk klasifikasi biner.
- **Hidden Layers**: Layer tersembunyi ditambahkan secara dinamis sesuai parameter.
- **Activation Function**: Fungsi aktivasi digunakan setelah setiap layer tersembunyi.
- **Output Layer**: Layer terakhir menggunakan 2 neuron (klasifikasi biner)
- **nn.Sequential**: Membuat jaringan secara berurutan.

➢ Membuat Fungsi Training dan Evaluasi

```
# Fungsi untuk melatih model
```

```
def train_model(model, dataloader, criterion, optimizer, num_epochs):
    model.train()  # Set model ke mode training
    for epoch in range(num_epochs):  # Iterasi setiap epoch
        running_loss = 0.0
        for inputs, labels in dataloader:
            optimizer.zero_grad()  # Reset gradient sebelumnya
            outputs = model(inputs)  # Forward pass
            loss = criterion(outputs, labels)  # Hitung loss
            loss.backward()  # Backward pass (menghitung gradien)
            optimizer.step()  # Update parameter
            running_loss += loss.item() * inputs.size(0)  # Total loss
        epoch_loss = running_loss / len(dataloader.dataset)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}")

# Fungsi untuk evaluasi model
def evaluate_model(model, dataloader):
    model.eval()  # Set model ke mode evaluasi
    predictions, targets = [], []
    with torch.no_grad():  # Tidak menghitung gradien
        for inputs, labels in dataloader:
            outputs = model(inputs)  # Forward pass
            _, preds = torch.max(outputs, 1)  # Mengambil prediksi kelas
            predictions.extend(preds.numpy())
            targets.extend(labels.numpy())
    accuracy = accuracy_score(targets, predictions)
    return accuracy
```

- **train_model** : Melatih model menggunakan forward pass, backward pass, dan parameter update, dan Menghitung loss untuk setiap batch dan mencetak loss rata-rata tiap epoch.
- **evaluate_model**: Mengevaluasi model menggunakan data testing dan menghitung akurasi.

➢ Experiment Grid Search

```
# Inisialisasi eksperimen
input_size = X_train.shape[1]  # Jumlah fitur input
results = []  # Menyimpan hasil eksperimen

# Looping eksperimen
for hidden_layers in [[4], [8, 4], [16, 8, 4]]:
    for activation_fn in [nn.ReLU(), nn.Sigmoid(), nn.Tanh()]:
        for lr in [0.1, 0.01, 0.001]:
            for epochs in [10, 50, 100]:
                for batch_size in [32, 64, 128]:
                    # Membuat model
```

```
            model = MLPClassifier(input_size=input_size, hidden_layers=hidden_layers,
activation_fn=activation_fn)

            # Membuat DataLoader
            train_loader = DataLoader(dataset_train, batch_size=batch_size, shuffle=True)
            test_loader = DataLoader(dataset_test, batch_size=batch_size, shuffle=False)

            # Inisialisasi loss function dan optimizer
            criterion = nn.CrossEntropyLoss()  # Digunakan untuk klasifikasi
            optimizer = torch.optim.Adam(model.parameters(), lr=lr)

            # Melatih model
            print(f"Training model with hidden_layers={hidden_layers}, activation_fn={activation_fn}, "
                  f"lr={lr}, epochs={epochs}, batch_size={batch_size}")
            train_model(model, train_loader, criterion, optimizer, epochs)

            # Evaluasi model
            acc = evaluate_model(model, test_loader)
            print(f"Accuracy: {acc:.4f}")

            # Menyimpan hasil eksperimen
            results.append({
                "hidden_layers": hidden_layers,
                "activation_fn": activation_fn.__class__.__name__,
                "lr": lr,
                "epochs": epochs,
                "batch_size": batch_size,
                "accuracy": acc
            })
```

- Loop mengeksplorasi berbagai kombinasi parameter (hidden layers, activation functions, learning rates, epochs, batch sizes).
- Model dilatih dengan kombinasi parameter tersebut dan akurasi disimpan.

➢ Analisis Hasil

```
# Mengonversi hasil eksperimen ke DataFrame untuk analisis
results_df = pd.DataFrame(results)

# Menampilkan hasil terbaik
best_result = results_df.loc[results_df['accuracy'].idxmax()]
print("\nBest Hyperparameter Combination:\n", best_result)

# Menyimpan hasil eksperimen ke file CSV
results_df.to_csv('/content/drive/MyDrive/Week 11/deep experiment_results.csv', index=False)
```

- Hasil eksperimen disimpan dalam DataFrame.
- Parameter terbaik dipilih berdasarkan akurasi tertinggi
- Hasil disimpan ke file CSV untuk referensi lebih lanjut.