Nama: Andi Cleopatra Maryam Jamila

Nim: 1103213071

Analisis RNN dan Deep RNN model Week 14

```
# Install PyTorch (skip if already installed)
!pip install torch torchvision torchaudio --quiet
```

```
# Import libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
# Load dataset
def load_data():
    url = "/content/sample_data/bank-full.csv"
    df = pd.read_csv(url, sep=';')

    # Preprocess data
    X = df.drop(columns=['y'])
    y = df['y']

    # Convert categorical to numerical
    X = pd.get_dummies(X)
    le = LabelEncoder()
    y = le.fit_transform(y)

    # Standardize features
    scaler = StandardScaler()
    X = scaler.fit_transform(X)

    return train_test_split(X, y, test_size=0.2, random_state=42)
```

Analisis:

Bertujuan untuk memuat, memproses, dan membagi dataset dalam rangka membangun model machine learning berbasis PyTorch. Dataset diambil dari file CSV yang berisi data pelanggan bank, dengan kolom target y yang mewakili variabel dependen (misalnya, apakah pelanggan menyetujui penawaran tertentu). Data preprocessing melibatkan penghapusan kolom target dari fitur input X, transformasi fitur kategorikal menjadi format numerik menggunakan **one-hot encoding** dengan pd.get_dummies(), serta pengubahan kolom target y menjadi nilai numerik menggunakan LabelEncoder. Selanjutnya, fitur X distandarkan menggunakan **StandardScaler** untuk memastikan skala data seragam, yang penting untuk algoritma berbasis gradien. Data yang telah diproses dibagi menjadi data pelatihan dan pengujian dengan rasio 80:20 menggunakan train_test_split, memastikan bahwa pengujian dilakukan

pada data yang tidak terlihat sebelumnya. Kode ini dirancang untuk membangun pipeline awal sebelum melatih model neural network dengan PyTorch.

```
# Create custom dataset class
class BankDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32).unsqueeze(1)  # Add sequence length dimension
        self.y = torch.tensor(y, dtype=torch.long)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

```
# Define RNN model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1, pooling='max'):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.pooling = pooling

        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        out, _ = self.rnn(x, h0)
        if self.pooling == 'max':
            out, _ = torch.max(out, 1)
        elif self.pooling == 'avg':
            out = torch.mean(out, 1)
        out = self.fc(out)
        return out
```

```
# Train function
def train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs=25):
    train_loader, val_loader = dataloaders
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)

        scheduler.step()

        val_loss = 0.0
        val_preds = []
        val_labels = []

        model.eval()
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item() * inputs.size(0)

                preds = torch.argmax(outputs, dim=1)
                val_preds.extend(preds.cpu().numpy())
                val_labels.extend(labels.cpu().numpy())

        train_loss = running_loss / len(train_loader.dataset)
        val_loss = val_loss / len(val_loader.dataset)
        val_acc = accuracy_score(val_labels, val_preds)

        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

    return model
```

Analisis:

membangun pipeline pelatihan model RNN (Recurrent Neural Network) menggunakan PyTorch untuk data sekuensial. **BankDataset** adalah kelas dataset kustom yang memformat data masukan menjadi tensor PyTorch dengan dimensi tambahan untuk panjang sekuensial, dan mengonversi label menjadi tipe data numerik. Model RNN didefinisikan dalam **RNNModel**, dengan parameter seperti ukuran input, ukuran hidden layer, jumlah lapisan, dan metode pooling (maksimum atau rata-rata) untuk mengurangi dimensi keluaran dari RNN sebelum diteruskan ke layer fully connected untuk klasifikasi. Fungsi **train_model** menangani proses pelatihan model dengan optimisasi berbasis gradien, validasi di setiap epoch, dan penyesuaian learning rate melalui scheduler. Selama pelatihan, model secara iteratif memperbarui bobotnya berdasarkan gradien, mengevaluasi kinerja pada data validasi, dan menghitung akurasi untuk memantau performa. Fungsionalitas ini dirancang untuk memungkinkan pelatihan model dengan fleksibilitas tinggi pada data bank yang sebelumnya diproses.

```python
# Main Experiment
if __name__ == "__main__":
    X_train, X_test, y_train, y_test = load_data()
    train_dataset = BankDataset(X_train, y_train)
    val_dataset = BankDataset(X_test, y_test)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

    dataloaders = (train_loader, val_loader)

    input_size = X_train.shape[1]
    output_size = len(np.unique(y_train))

    hidden_sizes = [16, 32, 64]
    poolings = ['max', 'avg']
    optimizers = {'SGD': optim.SGD, 'RMSProp': optim.RMSprop, 'Adam': optim.Adam}
    epochs_list = [5, 50, 100, 250, 350]

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    results = []

    for hidden_size in hidden_sizes:
        for pooling in poolings:
            for optimizer_name, optimizer_fn in optimizers.items():
                for num_epochs in epochs_list:
                    print(f"\nHidden Size: {hidden_size}, Pooling: {pooling}, Optimizer: {optimizer_name}, Epochs: {num_epochs}")

                    model = RNNModel(input_size, hidden_size, output_size, pooling=pooling)
                    criterion = nn.CrossEntropyLoss()
                    optimizer = optimizer_fn(model.parameters(), lr=0.01)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.5)

    trained_model = train_model(model, dataloaders, criterion, optimizer, scheduler, num_epochs)

    # Evaluate on test set
    test_preds = []
    test_labels = []

    trained_model.eval()
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = trained_model(inputs)
            preds = torch.argmax(outputs, dim=1)
            test_preds.extend(preds.cpu().numpy())
            test_labels.extend(labels.cpu().numpy())

    test_acc = accuracy_score(test_labels, test_preds)
    print(f"Test Accuracy: {test_acc:.4f}")

    results.append((hidden_size, pooling, optimizer_name, num_epochs, test_acc))
```

Output:

```
Epoch 99/100, Train Loss: 0.1521, Val Loss: 0.2604, Val Acc: 0.8964
Epoch 100/100, Train Loss: 0.1522, Val Loss: 0.2581, Val Acc: 0.8974
Test Accuracy: 0.8974

Hidden Size: 32, Pooling: avg, Optimizer: RMSProp, Epochs: 250
Epoch 1/250, Train Loss: 0.2352, Val Loss: 0.2389, Val Acc: 0.8936
Epoch 2/250, Train Loss: 0.2175, Val Loss: 0.2275, Val Acc: 0.9008
Epoch 3/250, Train Loss: 0.2117, Val Loss: 0.2191, Val Acc: 0.9019
Epoch 4/250, Train Loss: 0.2080, Val Loss: 0.2213, Val Acc: 0.9029
Epoch 5/250, Train Loss: 0.2043, Val Loss: 0.2356, Val Acc: 0.8942
Epoch 6/250, Train Loss: 0.2020, Val Loss: 0.2281, Val Acc: 0.9003
Epoch 7/250, Train Loss: 0.1985, Val Loss: 0.2235, Val Acc: 0.9045
Epoch 8/250, Train Loss: 0.1974, Val Loss: 0.2249, Val Acc: 0.8991
Epoch 9/250, Train Loss: 0.1948, Val Loss: 0.2241, Val Acc: 0.9012
Epoch 10/250, Train Loss: 0.1945, Val Loss: 0.2297, Val Acc: 0.8974
Epoch 11/250, Train Loss: 0.1927, Val Loss: 0.2323, Val Acc: 0.8993
Epoch 12/250, Train Loss: 0.1905, Val Loss: 0.2354, Val Acc: 0.8966
Epoch 13/250, Train Loss: 0.1894, Val Loss: 0.2345, Val Acc: 0.8990
Epoch 14/250, Train Loss: 0.1882, Val Loss: 0.2287, Val Acc: 0.9007
Epoch 15/250, Train Loss: 0.1873, Val Loss: 0.2261, Val Acc: 0.9047
Epoch 16/250, Train Loss: 0.1865, Val Loss: 0.2227, Val Acc: 0.9032
Epoch 17/250, Train Loss: 0.1848, Val Loss: 0.2278, Val Acc: 0.8996
Epoch 18/250, Train Loss: 0.1843, Val Loss: 0.2302, Val Acc: 0.9020
Epoch 19/250, Train Loss: 0.1836, Val Loss: 0.2335, Val Acc: 0.8965
Epoch 20/250, Train Loss: 0.1829, Val Loss: 0.2410, Val Acc: 0.8974
Epoch 21/250, Train Loss: 0.1823, Val Loss: 0.2417, Val Acc: 0.9008
Epoch 22/250, Train Loss: 0.1817, Val Loss: 0.2419, Val Acc: 0.8961
Epoch 23/250, Train Loss: 0.1802, Val Loss: 0.2311, Val Acc: 0.9021
Epoch 24/250, Train Loss: 0.1797, Val Loss: 0.2347, Val Acc: 0.9018
Epoch 25/250, Train Loss: 0.1794, Val Loss: 0.2322, Val Acc: 0.9030
Epoch 26/250, Train Loss: 0.1790, Val Loss: 0.2396, Val Acc: 0.9017
Epoch 27/250, Train Loss: 0.1780, Val Loss: 0.2506, Val Acc: 0.8989
Epoch 28/250, Train Loss: 0.1776, Val Loss: 0.2310, Val Acc: 0.9025
Epoch 29/250, Train Loss: 0.1771, Val Loss: 0.2463, Val Acc: 0.8993
Epoch 30/250, Train Loss: 0.1771, Val Loss: 0.2418, Val Acc: 0.8963
```

Analisis:

mengimplementasikan eksperimen utama untuk melatih dan mengevaluasi model RNN pada data bank yang telah diproses. Setelah membagi data menjadi training dan testing, data di-wrap ke dalam **BankDataset** dan diatur dalam batch menggunakan **DataLoader**. Eksperimen ini menjelajahi kombinasi hyperparameter seperti ukuran hidden layer (16, 32, 64), metode pooling (max, avg), jenis optimizer (SGD, RMSProp, Adam), dan jumlah epoch (5 hingga 350). Untuk setiap kombinasi, model RNN dilatih menggunakan **train_model** dengan loss function **CrossEntropyLoss**, optimizer yang dipilih, dan scheduler learning rate. Setelah pelatihan, model diuji pada data validasi untuk menghitung akurasi. Hasil eksperimen (kombinasi hyperparameter dan akurasi) disimpan untuk analisis selanjutnya. Pendekatan ini memungkinkan evaluasi sistematis terhadap konfigurasi terbaik untuk model RNN, berdasarkan akurasi pada dataset yang tersedia.