

Nama : Az – Zahra Chikal E

NIM : 1103213039

Kelas : TK-45-05

TUGAS WEEK 14 Bidirectional RNN

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
from tqdm import tqdm
Import "tqdm" could not be resolved from source
import matplotlib.pyplot as plt
import seaborn as sns
✓ 1.3s Python

# Load and preprocess data
data = pd.read_csv('bank-full.csv', delimiter=';')
data.head()
✓ 0.0s Python
```

Output:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	outcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

Analisis:

Kode diatas digunakan untuk membuat model machine learning menggunakan PyTorch. Data yang digunakan yaitu bernama `bank-full.csv`, yang berisi informasi tentang pelanggan bank. Data ini dimuat ke dalam program menggunakan Pandas dengan pemisah `;` karena format filenya menggunakan titik koma untuk memisahkan kolom. Untuk memahami isi dan struktur data, ditampilkan beberapa baris pertama dengan `data.head()`. Selanjutnya, data ini akan diproses dan dianalisis menggunakan alat bantu seperti NumPy, Scikit-learn, serta divisualisasikan menggunakan Matplotlib dan Seaborn sebelum digunakan untuk melatih model.

```
data.info()
✓ 0.0s Python

data.describe()
✓ 0.0s Python
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45211 entries, 0 to 45210
Data columns (total 17 columns):
#   Column      Non-Null Count  Dtype
---  ---
0    age      45211 non-null  int64
1    job      45211 non-null  object
2    marital  45211 non-null  object
3    education 45211 non-null  object
4    default  45211 non-null  object
5    balance  45211 non-null  int64
6    housing  45211 non-null  object
7    loan     45211 non-null  object
8    contact  45211 non-null  object
9    day      45211 non-null  int64
10   month    45211 non-null  object
11   duration 45211 non-null  int64
12   campaign 45211 non-null  int64
13   pdays    45211 non-null  int64
14   previous 45211 non-null  int64
15   poutcome 45211 non-null  object
16   y        45211 non-null  object
dtypes: int64(7), object(10)
memory usage: 5.9+ MB
```

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

Analisis:

Kode tersebut digunakan untuk mengetahui informasi terkait dataet. Dari code tersebut dapat kita tahu bahwa terdapat 16 kolom dengan jumlah data yaitu 45211. untuk datatype terdapat 2 yaitu int dan object. Fungsi `df.describe()` digunakan untuk memberikan ringkasan statistik dasar dari data numerik dalam sebuah DataFrame. Hasilnya mencakup informasi seperti jumlah data (count), rata-rata (mean), standar deviasi (std), nilai minimum (min), kuartil (25%, 50%, 75%), dan nilai maksimum (max) untuk setiap kolom numerik. Dengan menggunakan fungsi ini, kita dapat dengan cepat memahami karakteristik umum data, seperti distribusi nilai, rentang data, dan keberadaan kemungkinan nilai ekstrem.

```
print("\nTarget Variable Distribution:")
sns.countplot(x='y', data=data)
plt.title('Target Variable Distribution')
plt.show()
```

✓ 0.1s

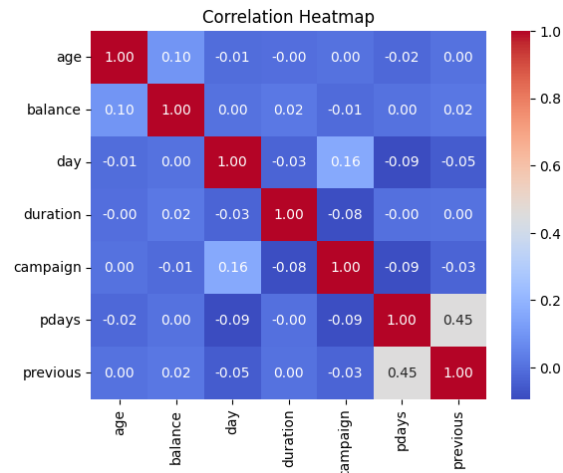
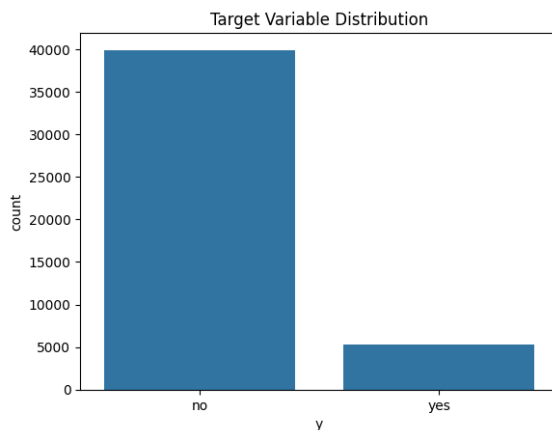
Python

```
print("\nCorrelation Heatmap:")
corr = data.select_dtypes(include=['float64', 'int64']).corr()
sns.heatmap(corr, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

✓ 0.1s

Python

Output:



Analisis:

Kode ini digunakan untuk memahami data dengan lebih mudah melalui grafik. Pertama, grafik batang dibuat untuk melihat distribusi nilai pada variabel target y, sehingga kita bisa mengetahui seberapa banyak setiap kategori muncul dalam data. Setelah itu, hubungan antara fitur numerik dalam dataset dihitung menggunakan korelasi, yang menunjukkan seberapa kuat dan arah hubungan antara dua variabel. Hasilnya ditampilkan dalam bentuk heatmap yang mempermudah kita melihat pola hubungan ini. Kedua langkah ini membantu kita memahami data sebelum melanjutkan ke analisis atau pemodelan lebih lanjut.

```
# preprocessing
X = data.drop('y', axis=1)
X = pd.get_dummies(X)
y = data['y'].apply(lambda x: 1 if x == 'yes' else 0)

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Dataset class
class CustomDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y.values, dtype=torch.float32)

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

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

train_dataset = CustomDataset(X_train, y_train)
test_dataset = CustomDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

Analisis:

Kode tersebut digunakan untuk memproses data dalam model machine learning berbasis PyTorch. Dataset dipisahkan menjadi fitur (X) dan target (y), di mana y diubah menjadi nilai numerik (1 untuk yes dan 0 untuk no). Fitur yang mengandung data kategorikal diubah menjadi bentuk numerik menggunakan one-hot encoding, lalu distandarisasi dengan StandardScaler untuk memastikan setiap fitur memiliki skala yang seragam. Data dibagi menjadi set pelatihan dan pengujian menggunakan rasio 80:20. Selanjutnya, kelas CustomDataset dibuat untuk mengelola data dalam format yang kompatibel dengan

PyTorch, mengubah data menjadi tensor, dan menyediakan metode untuk mengakses data berdasarkan indeks. Untuk efisiensi pelatihan, DataLoader digunakan untuk membuat batch data dengan ukuran 64, di mana data pelatihan diacak untuk meningkatkan generalisasi model. Proses ini memastikan data siap digunakan dalam pelatihan neural network.

```
# Define Bidirectional RNN model
class BiRNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, pooling='max'):
        super(BiRNN, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True, bidirectional=True)
        self.pooling = pooling
        self.fc = nn.Linear(hidden_size * 2, output_size)

    def forward(self, x):
        out, _ = self.rnn(x)
        if self.pooling == 'max':
            out = torch.max(out, dim=1).values
        elif self.pooling == 'avg':
            out = torch.mean(out, dim=1)
        out = self.fc(out)
        return out
```

✓ 0.0s Python

Analisis:

Kode tersebut mendefinisikan sebuah model *Bidirectional Recurrent Neural Network* (BiRNN) menggunakan PyTorch, yang dirancang untuk menangani data berbasis urutan, seperti teks atau data waktu. Model ini menggunakan arsitektur RNN bidirectional, memungkinkan jaringan mempelajari informasi dari urutan data maju dan mundur sekaligus, sehingga menghasilkan pemahaman konteks yang lebih baik. Input data pertama-tama diproses melalui lapisan RNN untuk menghasilkan output dari semua langkah waktu. Kemudian, output ini diringkas menggunakan salah satu metode pooling: *max pooling* untuk mengambil nilai maksimum, atau *average pooling* untuk menghitung rata-rata di sepanjang dimensi waktu. Setelah itu, hasil pooling diteruskan ke lapisan fully connected untuk menghasilkan prediksi akhir. Model ini dirancang fleksibel, dengan parameter yang dapat disesuaikan seperti ukuran input, jumlah unit tersembunyi, jumlah lapisan, dan metode pooling.

```

# Define training and evaluation loop
def train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader, test_loader):
    best_loss = float('inf')
    patience_counter = 0
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = model.to(device)

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        for x_batch, y_batch in tqdm(train_loader):
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            optimizer.zero_grad()
            outputs = model(X_batch.unsqueeze(1))
            loss = criterion(outputs.squeeze(), y_batch)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)

        model.eval()
        test_loss = 0
        correct = 0
        total = 0
        with torch.no_grad():
            for x_batch, y_batch in test_loader:
                X_batch, y_batch = X_batch.to(device), y_batch.to(device)
                outputs = model(X_batch.unsqueeze(1))
                loss = criterion(outputs.squeeze(), y_batch)
                test_loss += loss.item()
                predictions = torch.round(torch.sigmoid(outputs.squeeze()))
                correct += (predictions == y_batch).sum().item()
                total += y_batch.size(0)
            test_loss /= len(test_loader)
            accuracy = correct / total

        scheduler.step(test_loss)

        if test_loss < best_loss:
            best_loss = test_loss
            patience_counter = 0
        else:
            patience_counter += 1

        if patience_counter > early_stopper:
            print("Early stopping triggered")
            break

        print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Test Loss: {test_loss:.4f}, Accuracy: {accuracy:.4f}")

    return accuracy

```

✓ 0.0s

Python

Analisis:

Kode tersebut mendefinisikan fungsi untuk melatih dan mengevaluasi model deep learning dengan PyTorch. Fungsi ini mencakup mekanisme pelatihan model menggunakan data dari `train_loader`, di mana setiap batch diproses dengan menerapkan *forward pass*, menghitung loss dengan fungsi loss (criterion), melakukan *backpropagation*, dan memperbarui bobot model menggunakan optimizer. Setelah melatih model pada setiap epoch, model dievaluasi pada data pengujian dari `test_loader` dengan menghitung loss dan akurasi. Untuk meningkatkan efisiensi pelatihan, fungsi ini menggunakan scheduler untuk menyesuaikan *learning rate* berdasarkan kinerja model, serta menerapkan *early stopping* untuk menghentikan pelatihan lebih awal jika tidak ada perbaikan pada loss pengujian setelah beberapa epoch berturut-turut. Hasil pelatihan seperti train loss, test loss, dan akurasi dicetak setiap epoch untuk memantau kinerja model. Fungsi ini mengembalikan akurasi akhir setelah proses pelatihan selesai atau ketika *early stopping* dipicu.

```

# Experiment configurations
hidden_sizes = [16, 32, 64]
poolings = ['max', 'avg']
epochs_list = [5, 50, 100, 250, 350]
optimizers = ['SGD', 'RMSprop', 'Adam']
early_stopper = 10

# Run experiments and save results
results = []
for hidden_size in hidden_sizes:
    for pooling in poolings:
        for optimizer_name in optimizers:
            for num_epochs in epochs_list:
                print("Configuration: Hidden Size={hidden_size}, Pooling={pooling}, Optimizer={optimizer_name}, Epochs={num_epochs}")
                model = BiRNN(input_size=X_train.shape[1], hidden_size=hidden_size, num_layers=1, output_size=1, pooling=pooling)
                criterion = nn.BCEWithLogitsLoss()

                if optimizer_name == 'SGD':
                    optimizer = optim.SGD(model.parameters(), lr=0.01)
                elif optimizer_name == 'RMSprop':
                    optimizer = optim.RMSprop(model.parameters(), lr=0.01)
                elif optimizer_name == 'Adam':
                    optimizer = optim.Adam(model.parameters(), lr=0.01)

                scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=3, verbose=True)
                accuracy = train_model(model, optimizer, criterion, scheduler, num_epochs, early_stopper, train_loader, test_loader)

                results.append([
                    'Hidden Size': hidden_size,
                    'Pooling': pooling,
                    'Optimizer': optimizer_name,
                    'Epochs': num_epochs,
                    'Accuracy': accuracy
                ])

results_df = pd.DataFrame(results)
results_df.to_csv('experiment_bidirectional_rnn_results.csv', index=False)

```

Output:

```

=====
Configuration: Hidden Size=16, Pooling=max, Optimizer=SGD, Epochs=5
100%|#####| 566/566 [00:01<00:00, 300.25it/s]
Epoch 1/5, Train Loss: 0.4011, Test Loss: 0.3018, Accuracy: 0.8848
100%|#####| 566/566 [00:01<00:00, 529.41it/s]
Epoch 2/5, Train Loss: 0.2695, Test Loss: 0.2500, Accuracy: 0.8934
100%|#####| 566/566 [00:01<00:00, 516.42it/s]
Epoch 3/5, Train Loss: 0.2436, Test Loss: 0.2458, Accuracy: 0.8983
100%|#####| 566/566 [00:01<00:00, 501.86it/s]
Epoch 4/5, Train Loss: 0.2353, Test Loss: 0.2410, Accuracy: 0.8997
100%|#####| 566/566 [00:01<00:00, 505.43it/s]
Epoch 5/5, Train Loss: 0.2321, Test Loss: 0.2389, Accuracy: 0.8999
=====

```

Analisis:

Kode ini menjalankan serangkaian percobaan untuk menguji performa model *Bidirectional RNN* dengan berbagai pengaturan. Beberapa parameter diuji, seperti jumlah unit tersembunyi (*hidden size*), jenis metode pooling (max atau average), tipe optimizer (SGD, RMSprop, atau Adam), dan jumlah epoch pelatihan. Untuk setiap kombinasi parameter, model dilatih menggunakan data pelatihan dan dievaluasi pada data pengujian. Selain itu, digunakan scheduler untuk menyesuaikan *learning rate* berdasarkan performa model, serta mekanisme *early stopping* untuk menghentikan pelatihan jika tidak ada perbaikan pada loss validasi setelah beberapa epoch. Akurasi dari setiap pengaturan dicatat dan disimpan dalam file CSV agar hasilnya bisa dianalisis lebih lanjut. Dengan cara ini, kita dapat membandingkan performa model dengan berbagai kombinasi parameter dan menemukan pengaturan terbaik.

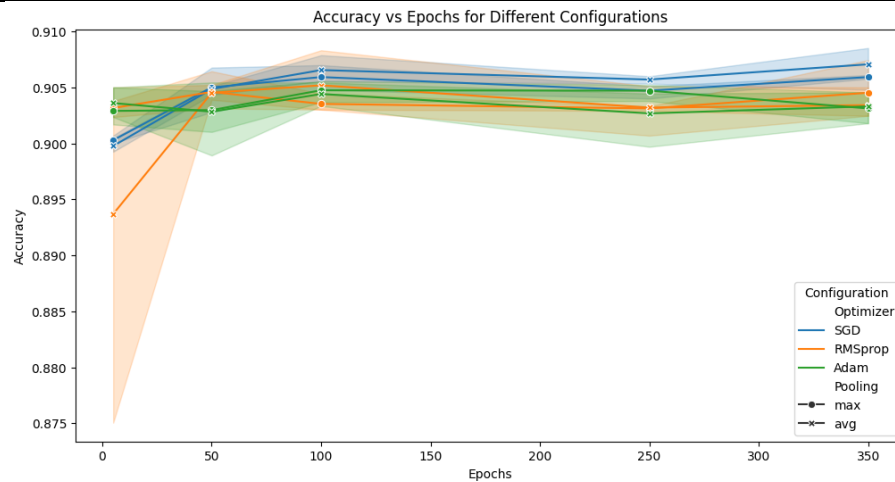
```

def visualize_results(results_df):
    plt.figure(figsize=(12, 6))
    sns.lineplot(data=results_df, x='Epochs', y='Accuracy', hue='Optimizer', style='Pooling', markers=True, dashes=)
    plt.title('Accuracy vs Epochs for Different Configurations')
    plt.ylabel('Accuracy')
    plt.xlabel('Epochs')
    plt.legend(title='Configuration', loc='lower right')
    plt.show()

visualize_results(results_df)

```

Output:



Analisis:

Kode ini digunakan untuk memvisualisasikan hasil percobaan model *Bidirectional RNN* dengan grafik. Fungsi `visualize_results` mengambil DataFrame hasil percobaan (`results_df`) dan membuat grafik garis menggunakan Seaborn. Grafik ini menunjukkan hubungan antara jumlah epoch pelatihan dan akurasi model. Garis-garis dalam grafik dikelompokkan berdasarkan jenis optimizer yang digunakan, dengan gaya garis dan tanda (*markers*) yang berbeda untuk membedakan metode pooling (max atau average). Grafik ini memberikan cara yang jelas untuk membandingkan performa model dengan berbagai konfigurasi, sehingga memudahkan untuk melihat pengaruh jumlah epoch, metode pooling, dan jenis optimizer terhadap akurasi. Hasil akhirnya ditampilkan dalam grafik yang memberikan wawasan visual untuk analisis lebih lanjut.

```
# Display top 10 as table
from tabulate import tabulate
top_10 = results_df.sort_values(by='Accuracy', ascending=False).head(10)
print("Top 10 Results:")
print(tabulate(top_10, headers='keys', tablefmt='pretty', showindex=False))
```

Output:

```
Top 10 Results:
+-----+-----+-----+-----+-----+
| Hidden Size | Pooling | Optimizer | Epochs | Accuracy |
+-----+-----+-----+-----+-----+
| 64 | avg | SGD | 350 | 0.9091009620701095 |
| 16 | avg | RMSprop | 100 | 0.9083268826716797 |
| 16 | avg | SGD | 100 | 0.9078845515868628 |
| 64 | max | RMSprop | 350 | 0.907442205020458 |
| 32 | avg | SGD | 350 | 0.9073316377308416 |
| 32 | max | SGD | 100 | 0.906998894172288 |
| 64 | max | SGD | 50 | 0.9067787238748203 |
| 16 | max | RMSprop | 50 | 0.9064469755612076 |
| 64 | avg | SGD | 100 | 0.9062258100187991 |
| 32 | max | SGD | 350 | 0.9061152272475949 |
+-----+-----+-----+-----+-----+
```

Analisis:

Kode tersebut menampilkan 10 hasil terbaik dari eksperimen model *Bidirectional RNN* berdasarkan akurasi tertinggi. Pertama, DataFrame `results_df` diurutkan berdasarkan kolom Accuracy dalam urutan menurun menggunakan `sort_values`. Sepuluh baris teratas dari hasil yang diurutkan diambil dengan `head(10)`. Untuk menyajikan hasil secara rapi, digunakan pustaka `tabulate`, yang membuat tabel dengan format yang lebih mudah dibaca. Tabel

menampilkan konfigurasi model seperti ukuran unit tersembunyi (*Hidden Size*), metode pooling (*Pooling*), jenis optimizer, jumlah epoch pelatihan, dan akurasi yang dicapai. Tabel ini memberikan gambaran jelas tentang kombinasi parameter yang menghasilkan performa terbaik dalam eksperimen.