Rizqy Asyraff Athallah

1103210158

Week 14

RNN Model

1. **Pra-Pemrosesan Data**

```
data = data.select_dtypes(include=[np.number]).dropna()
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
```

- Normalisasi sangat penting untuk algoritma pembelajaran mesin, terutama yang menggunakan gradien, agar fitur dengan skala berbeda dapat diproses secara setara.

2. **Membagi Dataset**

```
X = data_scaled[:, :-1]
y = data_scaled[:, -1]

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

- test_size=0.2 berarti 20% data digunakan untuk pengujian. random_state=42 memastikan hasil yang konsisten.

3. **Konversi ke Tensor**

```
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.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)
```

- Tensor adalah struktur data PyTorch yang digunakan untuk perhitungan pada GPU.

4. **Membuat DataLoader**

```
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

- batch_size=32: Ukuran batch untuk setiap iterasi pelatihan.
- shuffle=True: Mengacak data agar model tidak belajar urutan tertentu.

5. **Definisi Model RNN**

```
# Define RNN Model
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size,
pooling="max"):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
```

```
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.pooling = pooling
        if pooling == "max":
            self.pool = lambda x: torch.max(x, dim=1)[0]
        else:
            self.pool = lambda x: torch.mean(x, dim=1)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        rnn_out, _ = self.rnn(x)
        pooled_out = self.pool(rnn_out)
        output = self.fc(pooled_out)
        return output
```

- RNN digunakan untuk data berurutan.
- Pooling digunakan untuk merangkum informasi dari output RNN.

6. Parameter Experiment

```
hidden_sizes = [16, 32, 64]
pooling_types = ["max", "avg"]
epochs_list = [5, 50, 100, 250, 350]
optimizers = ["SGD", "RMSProp", "Adam"]
input_size = X_train.shape[1]
output_size = 1  # Regression output
```

-
- Parameter seperti ukuran hidden layer, teknik pooling, optimizer, dan jumlah epoch dibandingkan untuk menentukan kombinasi terbaik.

7. Pelatihan dan Evaluasi Model

```
# Run experiments
for hidden_size in hidden_sizes:
    for pooling in pooling_types:
        for optimizer_name in optimizers:
            for epochs in epochs_list:
                model = RNNModel(input_size, hidden_size, output_size,
pooling=pooling)
                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)

                criterion = nn.MSELoss()
                scheduler = optim.lr_scheduler.StepLR(optimizer,
step_size=10, gamma=0.1)

                # Training loop
```

```
                for epoch in range(epochs):
                    model.train()
                    for batch_X, batch_y in train_loader:
                        optimizer.zero_grad()
                        outputs = model(batch_X.unsqueeze(1))  # Add
sequence dimension
                        loss = criterion(outputs.squeeze(), batch_y)
                        loss.backward()
                        optimizer.step()
                    scheduler.step()

                # Evaluation
                model.eval()
                y_pred = []
                with torch.no_grad():
                    for batch_X, _ in test_loader:
                        outputs = model(batch_X.unsqueeze(1))
                        y_pred.extend(outputs.squeeze().tolist())

                # Store results
                results.append({
                    "hidden_size": hidden_size,
                    "pooling": pooling,
                    "optimizer": optimizer_name,
                    "epochs": epochs,
                    "mse_loss": criterion(torch.tensor(y_pred),
y_test_tensor).item()
                })
```

- Proses pelatihan mencakup optimasi, backward pass, dan pembaruan bobot.
- Scheduler digunakan untuk menyesuaikan learning rate.

8. Menyimpan dan Menganalisis Hasil

```
results_df = pd.DataFrame(results)
results_df.to_csv("experiment_results.csv", index=False)
```

- Data CSV ini dapat digunakan untuk membuat visualisasi atau laporan performa model berdasarkan parameter.