Rizqy Asyraff Athallah

1103210158

Week 14

Bidirectional RNN Model

1. **Pra-Pemrossan Data**

```
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)
X = data_scaled[:, :-1]
y = data_scaled[:, -1]
```

- **Normalisasi dengan StandardScaler**: Penting untuk mencegah skala fitur yang berbeda memengaruhi performa model.

2. **Training Dataset**

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- **test_size=0.2**: Memastikan cukup data untuk pengujian.
- **random_state=42**: Reproduksibilitas pembagian data.

3. **Membuat DataLoader**

```
# Create 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)
```

- **Batching**: Membagi data menjadi batch kecil (32 data per batch).
- **Shuffling**: Mengacak data pelatihan untuk mencegah model belajar pola urutan tertentu.

4. **Definisi Bidirectional RNN**

```
# Define Bidirectional RNN Model
class BidirectionalRNNModel(nn.Module):
```

```
    def __init__(self, input_size, hidden_size, output_size,
pooling="max"):
        super(BidirectionalRNNModel, self).__init__()
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True,
bidirectional=True)
        self.pooling = pooling
        self.fc = nn.Linear(hidden_size * 2, output_size)  # Multiply by 2
for bidirectional

    def forward(self, x):
        rnn_out, _ = self.rnn(x)
        if self.pooling == "max":
            x = torch.max(rnn_out, dim=1)[0]
        else:
            x = torch.mean(rnn_out, dim=1)
        return self.fc(x)
```

- Bidirectional:
  - Mengizinkan model membaca data dalam dua arah (ke depan dan ke belakang).
  - Berguna untuk menangkap informasi konteks yang lebih lengkap dalam data sekuensial.
- **Pooling**:
  - MaxPooling: Menangkap nilai maksimum dari setiap fitur.
  - AvgPooling: Mengambil rata-rata nilai untuk menghasilkan representasi yang lebih halus.

5. Fungsi Training

```
# Training function
def train_rnn(hidden_size, pooling, optimizer_name, epochs):
    model = BidirectionalRNNModel(input_size=X_train.shape[1],
hidden_size=hidden_size, output_size=1, pooling=pooling)
    optimizer = {"SGD": optim.SGD, "RMSProp": optim.RMSprop, "Adam":
optim.Adam}[optimizer_name](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()

    return model
```

- **Optimizers**:
  - SGD: Metode optimasi dasar yang lambat tetapi stabil.
  - RMSProp: Cepat untuk data sekuensial.
  - Adam: Kombinasi terbaik dari SGD dan RMSProp.
- **Scheduler**: Menyesuaikan learning rate selama pelatihan untuk meningkatkan konvergensi.

6. Experiment Parameter

```
for hidden_size in hidden_sizes:
    for pooling in pooling_types:
        for optimizer_name in optimizers:
            for epochs in epochs_list:
                model = train_rnn(hidden_size, pooling, optimizer_name,
epochs)
                model.eval()
                with torch.no_grad():
                    predictions = []
                    for batch_X, _ in test_loader:
                        pred = model(batch_X.unsqueeze(1))
                        predictions.extend(pred.squeeze().tolist())
                mse_loss = nn.MSELoss()(torch.tensor(predictions),
y_test_tensor).item()
                results.append({
```

```
                    "hidden_size": hidden_size,
                    "pooling": pooling,
                    "optimizer": optimizer_name,
                    "epochs": epochs,
                    "mse_loss": mse_loss
                })
```

- **Hidden Size**: Mengontrol kapasitas jaringan.
- **Pooling**: Membandingkan efek MaxPooling vs AvgPooling.
- **Optimizer dan Epoch**: Menguji stabilitas dan kecepatan konvergensi model.

7. Menyimpan Hasil

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

- File ini dapat digunakan untuk analisis performa model lainnya