Prediksi Kurs Rupiah terhadap Yen menggunakan RNN dan LSTM

Muhammad Farhan Arya Wicaksono
NRP: 5054231011

Laporan Tugas Mata Kuliah *Deep Learning*

1 Pendahuluan

Laporan ini membahas dan membandingkan hasil prediksi kurs Rupiah terhadap Yen dengan menggunakan model **RNN (Recurrent Neural Network)** dan **LSTM (Long Short-Term Memory)**. Model *Deep Learning* berbasis sekuenes ini digunakan untuk menangkap pola dalam data time series dan dapat memperkirakan nilai tukar di masa depan.

2 Metodologi

2.1 Data dan Preprocessing

Data diunduh dari `https://satudata.kemendag.go.id/data-informasi/perdagangan-dalam-negeri/nilai-tukar` berupa kurs dari beberapa negara di dunia dari periode 2001 hingga sekarang tahun 2025, dengan data kurs yang disajikan dalam format bulanan.

Data ini mencakup berbagai mata uang, seperti USD, JPY, GBP, dan lain-lain, yang digunakan untuk analisis perbandingan nilai tukar terhadap Rupiah. Namun, untuk keperluan analisis ini, hanya data nilai tukar JPY (Jepang) yang diambil sebagai subset data. Penggabungan kolom "Tahun" dan "Bulan" menjadi satu kolom waktu, serta pemilihan fitur mata uang yang relevan untuk analisis lebih lanjut.

- **Training:** Januari 2001 - Desember 2022
- **Testing:** Januari 2023 - Desember 2023

Preprocessing mencakup:

- Normalisasi menggunakan MinMaxScaler
- Pembuatan sequence (window = 12 bulan)

2.2 Arsitektur Model

Pembuatan model dilakukan dengan mencoba berbagai macam konfigurasi **(tuning)** yang berbeda-beda dari jumlah **neuron**, **layer**, **dropout**, **epoch**, **optimizer**, dan **learning rate**.

Proses tuning ini diterapkan pada 2 jenis model, yaitu **RNN** dan **LSTM**, dengan konfigurasi yang serupa untuk membandingkan performa masing-masing agar tidak bias.

Model	Hidden Sizes	Dropout	Learning Rate	Epochs	Optimizer
1	[16]	0.1	0.001	10	Adam
2	[32]	0.1	0.0008	15	Adam
3	[64, 128]	0.25	0.0005	25	Adam
4	[64, 128, 256]	0.3	0.0003	30	RMSprop
5	[128, 256]	0.35	0.0002	35	AdamW
6	[128, 256, 512]	0.4	0.0001	40	Adam
7	[256, 512, 1024]	0.45	0.00005	50	AdamW
8	[512, 1024, 2048]	0.5	0.00001	60	Adam
9	[128, 256, 512]	0.3	0.0003	30	SGD
10	[64, 128, 256, 512]	0.4	0.0001	50	Adam

Tabel 1: Konfigurasi Model yang Digunakan dalam Eksperimen

3 Implementasi Kode

3.1 Import Library

```
import torch
import torch.nn as nn
```

```
import torch.optim as optim
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from datetime import datetime
from sklearn.metrics import mean_squared_error
from tqdm import tqdm
import torch.optim as optim
```

3.2 Formatting Data

```
data = pd.read_excel('kurs_rupiah.xlsx')
hasil = []

current_year = 2025  # Inisialisasi tahun awal

def ubah_format(listData):
    global current_year
    if isinstance(listData[0], int) or (isinstance(listData[0], str) and
   listData[0].isdigit()):
        current_year = str(listData[0])  # Perbarui tahun
    else:
        hasil.append([str(current_year) + "-" + listData[0], str(listData
   [1])])

for data in JAPAN.iloc[1:].values:
  ubah_format(data)

clean = pd.DataFrame(hasil, columns=['Tahun', 'JPY'])
clean['JPY'] = clean['JPY'].str.replace(',', '.').astype(float)
clean.head()

output:

Tahun JPY
0 2025-Januari  10.52363
1 2024-Desember 10.23625
2 2024-November 10.45301
3 2024-Oktober  10.25989
4 2024-September  10.56644
... ... ...
284 2001-Mei  9.21733
285 2001-April  9.42066
286 2001-Maret  8.37000
287 2001-Februari 8.45297
288 2001-Januari  8.13149
289 rows x 2 columns

clean.to_csv('jpy.csv', index = False)
```

3.3 Preparing and Preprocessing Data

```
# Set CUDA
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

# Load data
df = pd.read_csv('jpy.csv')
df.head()

# Information about the data
df.info()

# Convert month names in Indonesian to English
bulan_indo_ke_eng = {
    "Januari": "January", "Februari": "February", "Maret": "March",
    "April": "April", "Mei": "May", "Juni": "June",
    "Juli": "July", "Agustus": "August", "September": "September",
    "Oktober": "October", "November": "November", "Desember": "December"
}

# Function to convert date string to datetime format
def convert_to_datetime(date_str):
    tahun, bulan_indo = date_str.split('-')
    bulan_eng = bulan_indo_ke_eng[bulan_indo]
    return datetime.strptime(f"{tahun}-{bulan_eng}", "%Y-%B")

# Apply conversion to DataFrame
df["tanggal_datetime"] = df["Tahun"].apply(convert_to_datetime)

# Convert to datetime
df['tanggal_datetime'] = pd.to_datetime(df['tanggal_datetime'])
df.set_index('tanggal_datetime', inplace=True)

# Sort index (from year 2001 to 2023)
df = df.sort_index()

# Normalize data
scaler = MinMaxScaler()
df['JPY'] = scaler.fit_transform(df[['JPY']])

# Training from 2001 to 2022-12-31
train_data = df.loc['2001-01-01':'2022-12-01']

# Test data from 2022-01-01 to 2023-12-01
test_data = df.loc['2022-01-01':'2023-12-01']

# Function to create sequences for model training
sequence_length = 12

def create_sequences(data, sequence_length):
    sequences = []
```

```
    labels = []
    for i in range(len(data) - sequence_length):
        seq = data[i:i+sequence_length]
        label = data[i+sequence_length]
        sequences.append(seq)
        labels.append(label)
    return np.array(sequences), np.array(labels)

train_sequences, train_labels = create_sequences(train_data['JPY'].values, sequence_length)
test_sequences, test_labels = create_sequences(test_data['JPY'].values, sequence_length)

# Convert data to PyTorch tensors
X_train = torch.tensor(train_sequences, dtype=torch.float32).unsqueeze(-1).to(device)
y_train = torch.tensor(train_labels, dtype=torch.float32).unsqueeze(-1).to(device)
X_test = torch.tensor(test_sequences, dtype=torch.float32).unsqueeze(-1).to(device)
y_test = torch.tensor(test_labels, dtype=torch.float32).unsqueeze(-1).to(device)
```

3.4 Pembuatan Model

```
# Define RNN Model
class CustomRNN(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, dropout):
        super(CustomRNN, self).__init__()
        self.num_layers = len(hidden_sizes)
        self.rnn_layers = nn.ModuleList()
        self.rnn_layers.append(nn.RNN(input_size, hidden_sizes[0], batch_first=True))

        for i in range(len(hidden_sizes) - 1):
            self.rnn_layers.append(nn.RNN(hidden_sizes[i], hidden_sizes[i + 1], batch_first=True))

        self.fc = nn.Linear(hidden_sizes[-1], output_size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        for rnn in self.rnn_layers:
            x, _ = rnn(x)
            x = self.dropout(x)
        out = self.fc(x[:, -1, :])
        return out

# Define LSTM Model
class CustomLSTM(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, dropout):
        super(CustomLSTM, self).__init__()
```

```
        self.num_layers = len(hidden_sizes)
        self.lstm_layers = nn.ModuleList()
        self.lstm_layers.append(nn.LSTM(input_size, hidden_sizes[0],
    batch_first=True))

        for i in range(len(hidden_sizes) - 1):
            self.lstm_layers.append(nn.LSTM(hidden_sizes[i], hidden_sizes[
    i + 1], batch_first=True))

        self.fc = nn.Linear(hidden_sizes[-1], output_size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        for lstm in self.lstm_layers:
            x, _ = lstm(x)
            x = self.dropout(x)
        out = self.fc(x[:, -1, :])
        return out
```

3.5 Model Konfigurasi and Training

```
configs = [
    # Simple Model (Baseline)
    {"hidden_sizes": [16], "dropout": 0.1, "lr": 0.001, "epochs": 10, "
    optimizer": "Adam"},
    {"hidden_sizes": [32], "dropout": 0.1, "lr": 0.0008, "epochs": 15, "
    optimizer": "Adam"},

    # Mid-sized Model
    {"hidden_sizes": [64, 128], "dropout": 0.25, "lr": 0.0005, "epochs":
    25, "optimizer": "Adam"},
    {"hidden_sizes": [64, 128, 256], "dropout": 0.3, "lr": 0.0003, "epochs
    ": 30, "optimizer": "RMSprop"},
    {"hidden_sizes": [128, 256], "dropout": 0.35, "lr": 0.0002, "epochs":
    35, "optimizer": "AdamW"},

    # Complex Model
    {"hidden_sizes": [128, 256, 512], "dropout": 0.4, "lr": 0.0001, "
    epochs": 40, "optimizer": "Adam"},
    {"hidden_sizes": [256, 512, 1024], "dropout": 0.45, "lr": 0.00005, "
    epochs": 50, "optimizer": "AdamW"},
    {"hidden_sizes": [512, 1024, 2048], "dropout": 0.5, "lr": 0.00001, "
    epochs": 60, "optimizer": "Adam"},

    # Additional Model
    {"hidden_sizes": [128, 256, 512], "dropout": 0.3, "lr": 0.0003, "
    epochs": 30, "optimizer": "SGD"},
    {"hidden_sizes": [64, 128, 256, 512], "dropout": 0.4, "lr": 0.0001, "
    epochs": 50, "optimizer": "Adam"}
]
```

3.6 Model Implementasi and Training Loop

```
for idx, config in enumerate(configs):
    loss_rnn_list = []
    loss_lstm_list = []
    loss_gru_list = []

    hidden_sizes = config["hidden_sizes"]
    dropout = config["dropout"]
    lr = config["lr"]
    epochs = config["epochs"]
    optimizer_name = config["optimizer"]

    try:
      model_rnn = CustomRNN(input_size, hidden_sizes, output_size, dropout
    ).to(device)
      model_lstm = CustomLSTM(input_size, hidden_sizes, output_size,
    dropout).to(device)
    except:
      model_rnn = CustomRNN(input_size, hidden_sizes, output_size, dropout
    )
      model_lstm = CustomLSTM(input_size, hidden_sizes, output_size,
    dropout)

    optimizer_rnn = optim.Adam(model_rnn.parameters(), lr=lr) if
    optimizer_name == "Adam" else optim.RMSprop(model_rnn.parameters(), lr=
    lr)
    optimizer_lstm = optim.Adam(model_lstm.parameters(), lr=lr) if
    optimizer_name == "Adam" else optim.RMSprop(model_lstm.parameters(), lr
    =lr)

    criterion = nn.MSELoss()

    for epoch in tqdm(range(epochs), desc=f"Model {idx+1} Training
    Progress"):
        optimizer_rnn.zero_grad()
        optimizer_lstm.zero_grad()

        output_rnn = model_rnn(X_train)
        output_lstm = model_lstm(X_train)

        loss_rnn = criterion(output_rnn, y_train)
        loss_lstm = criterion(output_lstm, y_train)

        loss_rnn.backward()
        loss_lstm.backward()

        optimizer_rnn.step()
        optimizer_lstm.step()

        loss_rnn_list.append(loss_rnn.item())
        loss_lstm_list.append(loss_lstm.item())

        if (epoch + 1) % 10 == 0 or epoch == 0:
```

```
            print(f"Epoch {epoch+1}/{epochs} -> RNN Loss: {loss_rnn.item()
    :.5f}, LSTM Loss: {loss_lstm.item():.5f}")
```

3.7 Evaluasi dan hasil

```
model_rnn.eval()
model_lstm.eval()

with torch.no_grad():
    pred_rnn = model_rnn(X_test)
    pred_lstm = model_lstm(X_test)

mse_rnn = criterion(pred_rnn, y_test).item()
mse_lstm = criterion(pred_lstm, y_test).item()

avg_loss_rnn = sum(loss_rnn_list) / len(loss_rnn_list)
avg_loss_lstm = sum(loss_lstm_list) / len(loss_lstm_list)

results.append([
    idx+1, len(hidden_sizes), hidden_sizes, dropout, lr, epochs,
    optimizer_name,
    avg_loss_rnn, avg_loss_lstm, mse_rnn, mse_lstm
])
```

3.8 Visualisasi and Plotting

```
# Inverse transform to get original scale
y_test_inv = scaler.inverse_transform(y_test.cpu().reshape(-1, 1))
y_pred_rnn_inv = scaler.inverse_transform(pred_rnn.cpu().reshape(-1, 1))
y_pred_lstm_inv = scaler.inverse_transform(pred_lstm.cpu().reshape(-1, 1))

plt.figure(figsize=(12, 6))

test_index_plot = test_index[12:]  # Adjust index for plotting

# Membuat subplot 2x3 untuk prediksi dan loss
fig, axes = plt.subplots(2, 2, figsize=(18, 12))

# Plot untuk RNN - Prediksi
axes[0, 0].plot(test_index_plot, y_test_inv, label='Actual', color='black'
    )
axes[0, 0].plot(test_index_plot, y_pred_rnn_inv, label='RNN Prediction',
    color='blue')
axes[0, 0].set_title('RNN Predictions')
axes[0, 0].set_xlabel('Time')
axes[0, 0].set_ylabel('JPY Value')
axes[0, 0].legend()

# Plot untuk LSTM - Prediksi
```

```
axes[0, 1].plot(test_index_plot, y_test_inv, label='Actual', color='black')
axes[0, 1].plot(test_index_plot, y_pred_lstm_inv, label='LSTM Prediction', color='red')
axes[0, 1].set_title('LSTM Predictions')
axes[0, 1].set_xlabel('Time')
axes[0, 1].set_ylabel('JPY Value')
axes[0, 1].legend()

# Plot untuk MSE
axes[1, 0].plot(loss_rnn_list, label='RNN Loss', color='blue')
axes[1, 0].plot(loss_lstm_list, label='LSTM Loss', color='red')
axes[1, 0].set_title('Loss per Epoch')
axes[1, 0].set_xlabel('Epochs')
axes[1, 0].set_ylabel('Loss')
axes[1, 0].legend()
```

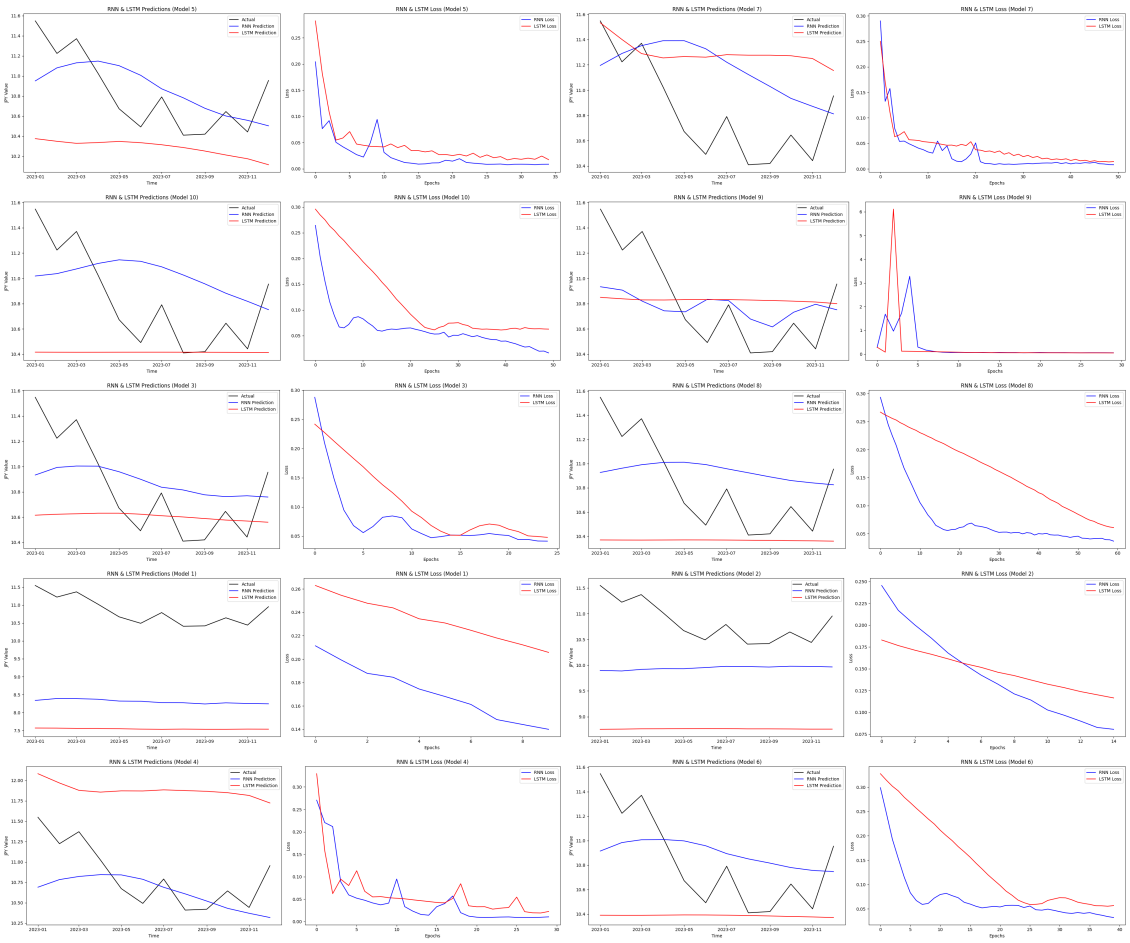
4 Perbandingan MSE

	Model No	Num Layers	Hidden Size	Dropout	Learning Rate	Epochs	Optimizer	Final RNN Loss	Final LSTM Loss	Test MSE RNN	Test MSE LSTM
0	1	1	[16]	0.10	0.00100	10	Adam	0.367447	0.186677	0.258970	0.117861
1	2	1	[32]	0.10	0.00080	15	Adam	0.081458	0.439577	0.005583	0.329272
2	3	2	[64, 128]	0.25	0.00050	25	Adam	0.076209	0.148349	0.002039	0.010070
3	4	3	[64, 128, 256]	0.30	0.00030	30	RMSprop	0.069834	0.059730	0.002051	0.003855
4	5	2	[128, 256]	0.35	0.00020	35	AdamW	0.034255	0.051893	0.002384	0.007118
5	6	3	[128, 256, 512]	0.40	0.00010	40	Adam	0.065302	0.116863	0.002220	0.006579
6	7	3	[256, 512, 1024]	0.45	0.00005	50	AdamW	0.031890	0.046480	0.005146	0.006920
7	8	3	[512, 1024, 2048]	0.50	0.00001	60	Adam	0.072905	0.172188	0.002016	0.005620
8	9	3	[128, 256, 512]	0.30	0.00030	30	SGD	0.328725	0.312881	0.002171	0.002110
9	10	4	[64, 128, 256, 512]	0.40	0.00010	50	Adam	0.065889	0.102702	0.002120	0.003878

Gambar 1: Hasil Loss dan MSE dari tiap model

5 Visualisasi Prediksi

lihat pada gambar 2



Gambar 2: Hasil Loss dan MSE dari tiap model

6 Analisis

6.1 Perbandingan Final RNN Loss vs Final LSTM Loss

- LSTM umumnya menunjukkan nilai loss yang lebih rendah dibandingkan RNN, kecuali pada model 1 dan 7, di mana RNN lebih baik.
- Secara keseluruhan, LSTM menunjukkan performa yang lebih stabil dalam mengurangi loss dibandingkan RNN.

6.2 Pengaruh Jumlah Layer dan Ukuran Lapisan Tersembunyi

- Model dengan 2 hingga 3 lapisan (Model 2 dan Model 3) memberikan kinerja terbaik, dengan MSE lebih rendah.
- Ukuran lapisan yang lebih besar tidak selalu menghasilkan performa lebih baik, terlihat

pada Model 8 dengan MSE lebih tinggi meskipun memiliki ukuran lapisan besar.

6.3 Pengaruh Dropout dan Learning Rate

- Dropout yang lebih tinggi, seperti pada Model 7 dan Model 8, membantu kinerja LSTM meskipun tidak selalu menguntungkan untuk RNN.
- Learning rate rendah (seperti pada Model 2 dan Model 5) cenderung memberikan hasil baik, terutama pada LSTM.

6.4 Perbandingan Test MSE RNN vs Test MSE LSTM

- LSTM memiliki MSE yang lebih rendah dibandingkan RNN pada sebagian besar model, yang menunjukkan prediksi yang lebih baik.
- Model dengan optimizer AdamW menunjukkan hasil baik pada kedua jenis model.

6.5 Perbandingan Optimizer

- AdamW menunjukkan performa terbaik dengan MSE yang lebih rendah dibandingkan dengan optimizer lain seperti SGD.
- Adam dan AdamW efektif dalam mengurangi loss dan MSE, sementara SGD kurang efektif.

6.6 Pengaruh Epochs

- Model dengan jumlah epoch lebih tinggi, seperti Model 6 dan Model 7, cenderung memberikan hasil yang lebih baik pada LSTM, meskipun tidak selalu berpengaruh besar pada RNN.
- Epoch yang lebih banyak memungkinkan model untuk belajar lebih banyak, namun berisiko overfitting jika tidak diimbangi dengan teknik regulasi yang tepat.

6.7 Model Terbaik

- Model dengan 5 lapisan dan dua hidden layer ([128, 256]) menggunakan AdamW dengan dropout 0.35 pada epoch ke-35 menghasilkan MSE uji terendah.
- LSTM secara konsisten memberikan hasil yang lebih baik dibandingkan RNN.

7 Kesimpulan

- **LSTM lebih unggul** dalam hal akurasi prediksi dan stabilitas dibandingkan RNN.
- **AdamW** adalah optimizer terbaik dalam mengurangi MSE.
- hasil terbaik adalah menggunakan **LSTM dengan AdamW** dan konfigurasi model yang lebih dalam (lebih banyak deep layer).