# PMDS603P Deep Learning Lab Experiment 9

September 2025

## 1 Work to do today

Note: Make a single PDF file of the work you are doing in a Jupyter notebook. Upload with the proper format. Please mention your name and roll no properly with the Experiment number on the first page of your submission.

Today we will look at how we can define a simple RNN model to predict the Gold price which is in fact, a data which is having some historical trends and other relevant information in it. A best choice to handle this kind of data is RNN (Recurrent Neural Networks) model.

**Question 1**: First, we will try to load the dataset and do the pre-processing part. From Github i have taken this data set which has information of monthly gold price till a latest date.

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense
from keras.callbacks import EarlyStopping

np.random.seed(42)
df = pd.read_csv("gold_price.csv")
print(df.head())
data = df["Price"].values.reshape(-1, 1)

scaler = MinMaxScaler()
data scaled = scaler.fit transform(data)
```

Now we can prepare the training, validation, and testing sequences of data.

```
train_size = int(len(data_scaled) * 0.8)
train_data = data_scaled[:train_size]
test_data = data_scaled[train_size:]

def create_sequences(data, seq_length):
    sequences = []
    for i in range(len(data) - seq_length):
        sequences.append(data[i:i+seq_length])
    return np.array(sequences)

seq_length = 10
train_sequences = create_sequences(train_data, seq_length)
test_sequences = create_sequences(test_data, seq_length)
```

Next, we will split the sequences into inputs and targets. And do the validation split and use it for further training.

```
# Split sequences into inputs and targets
X_train_full, y_train_full = train_sequences[:, :-1], train_sequences[:, -1]
X_test, y_test = test_sequences[:, :-1], test_sequences[:, -1]

val_fraction = 0.2
val_size = int(len(X_train_full) * val_fraction)

X_val = X_train_full[-val_size:]
y_val = y_train_full[-val_size:]

X_train = X_train_full[:-val_size]
y_train = y_train_full[:-val_size]
```

Further, we can define the model and compile it and fit the model.

```
model = Sequential()
model.add(SimpleRNN(64, activation='tanh', input_shape=(seq_length-1, 1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_data=(X_val, y_val),
    verbose=1,
    callbacks=[early_stop]
)
```

Find the predictions for your model and plot the actual data and the predicted data to see your model. Also we can see the mean squared error and other performance metrics to evaluate the model performance.

```
y_pred = model.predict(X_test)

y_test_orig = scaler.inverse_transform(y_test)
y_pred_orig = scaler.inverse_transform(y_pred)

rootmse = np.sqrt(mean_squared_error(y_test_orig, y_pred_orig))
print('Root Mean Squared Error:', rootmse)

# Plot True vs Predicted
plt.figure(figsize=(8, 4))
plt.plot(y_test_orig, label='True')
plt.plot(y_pred_orig, label='Predicted')
plt.xlabel('Time')
plt.ylabel('Gold Price')
plt.legend()
plt.show()
```

**Question 2:** Now the previous model we have seen takes an input sequence of length 9 and predicts the next days gold price. if you want to have a model which takes 10 days of data and predict the next ten days of gold price (lets say), then the same model can be modified to get output at every time step by introducing a timedistributed dense layer and also keeping the return_sequences parameter = True. But here in order to attempt this problem you need to prepare your input and output sequences accordingly to the model so we can perform these to prepare the sequence initially.

First you can import necessary layers, use

**from keras.layers import SimpleRNN, Dense, TimeDistributed**

Next, we will prepare our training sequences of data.

```
def create_sequences(data, input_len=10, output_len=10):
    X, y = [], []
    for i in range(len(data) - input_len - output_len + 1):
        X.append(data[i:i+input_len])
        y.append(data[i+input_len:i+input_len+output_len])
    return np.array(X), np.array(y)

input_len = 10
output_len = 10

X_train_full, y_train_full = create_sequences(train_data, input_len, output_len)
X_test, y_test = create_sequences(test_data, input_len, output_len)
```

Then we have validation split.

```
val_fraction = 0.2
val_size = int(len(X_train_full) * val_fraction)

X_val = X_train_full[-val_size:]
y_val = y_train_full[-val_size:]

X_train = X_train_full[:-val_size]
y_train = y_train_full[:-val_size]
```

Further the model can be defined and fitted Finally do the predictions and check the

```
model = Sequential()
model.add(SimpleRNN(64, activation='tanh', return_sequences=True, input_shape=(input_len, 1)))
model.add(TimeDistributed(Dense(1)))  # predicts for each timestep

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_data=(X_val, y_val),
    verbose=1,
    callbacks=[early_stop]
)
```

results.

**Question 3.** Next if we can check the same problem with a deep RNN. So we will introduce more layers and see how we can proceed. We will try for a many to one RNN model, where we will take 10 days continuous data and predict the next day data using deep RNN model. That is we are going to add few layers in the model.

Now this is an example of an RNN model with enough layers with other dense layers attached. try to fit this model and see the errror and performance.

```
model = Sequential()
model.add(SimpleRNN(128, activation='tanh', return_sequences=True, input_shape=(seq_length-1, 1)))
model.add(SimpleRNN(64, activation='tanh', return_sequences=True))
model.add(SimpleRNN(32, activation='tanh'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
```

A stacked RNN with two layers is something like this when unfolded in time.