CSE 655 / Deep Learning and Applications

Project Report

Project Name: PJM Hourly Energy Consumption

Student: Bülent Karadeniz / 225023016010

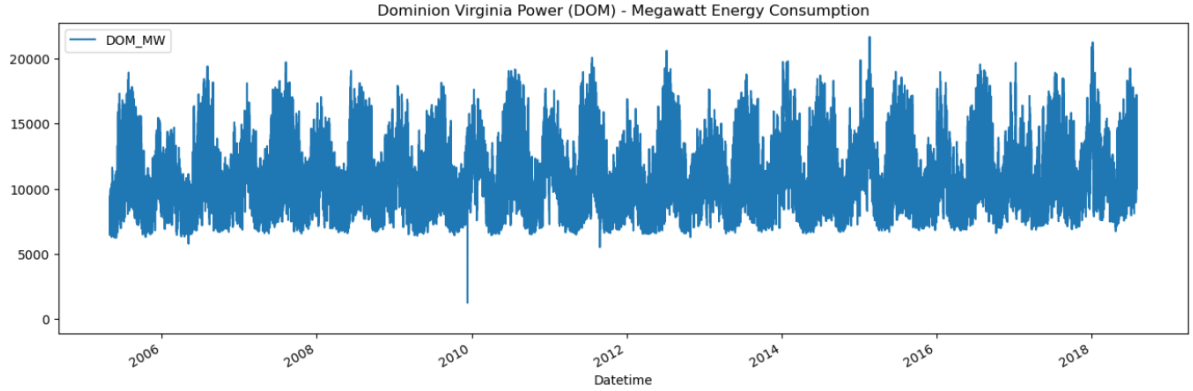
Contex

1. Introduction

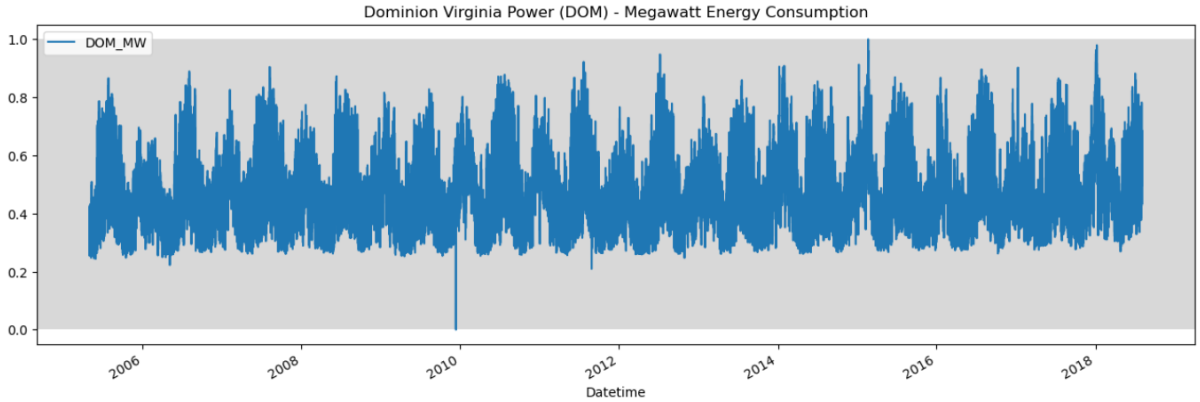
- PJM Interconnection LLC (PJM) is a regional transmission organization (RTO) in the United States. It is part of the Eastern Interconnection grid operating an electric transmission system serving all or parts of Delaware, Illinois, Indiana, Kentucky, Maryland, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania, Tennessee, Virginia, West Virginia, and the District of Columbia.
- PJM Interconnection's operation of the high-voltage power grid, wholesale electricity markets and its long-term planning process provide significant value to the 65 million people in the region it serves. PJM operations, markets and planning result in annual savings of $3.2−4 billion. These savings represent the vital functions that PJM provides and that lead to less cost to consumers:
- The hourly power consumption data comes from PJM's website and are in megawatts (MW).
- The regions have changed over the years so data may only appear for certain dates per region.
- There are 13 regions. In this notebook we will continue using data from Dominion Virginia Power (DOM).
- Developing an RNN and LSTM energy consumption forecasting model and regularizing the model parameters using data from PJM electricity distribution company.
- The main objective of this project is to predict the energy needs of the company for the region in the coming years with the existing data set.

2. Data Collection and Preprocessing

- The data set was obtained from Kaggle. Dominion Virginia Power (DOM) data was used. Data file format csv was used in the model.(DOM_hourly.csv)
- The data set consists of datetime (hourly) and consumption (megawatt) columns.
- Data covers hourly energy consumption between 2005 and 2018. There are also no null values in the data set.
- Data columns (total 2 columns):
  # Column Non-Null Count Dtype
  0 Datetime 116189 non-null object
  1 DOM_MW 116189 non-null float64
- In the preprocessing process, we first made a datetime index in the data set.
- Now let's observe our data set on the graph



- We did a normalization process. We did normalization, we pulled our data between 0 -1 values.



- 2017-02-13 after this date we will choose the test set

- Prepare data for trainin the RNN and LSTM models

- With the following function block, let's set our data set as training and test data set in a model appropriate way

```
def load_data(data, seq_len):
    X_train = []
    y_train = []

    for i in range(seq_len, len(data)):
        X_train.append(data.iloc[i-seq_len : i, 0])
        y_train.append(data.iloc[i, 0])

    # last 6189 days are going to be used in test
    X_test = X_train[110000:]
    y_test = y_train[110000:]

    # first 110000 days are going to be used in training
    X_train = X_train[:110000]
    y_train = y_train[:110000]

    # convert to numpy array
    X_train = np.array(X_train)
    y_train = np.array(y_train)

    X_test = np.array(X_test)
    y_test = np.array(y_test)

    # reshape data to input into RNN&LSTM models
    X_train = np.reshape(X_train, (110000, seq_len, 1))

    X_test = np.reshape(X_test, (X_test.shape[0], seq_len, 1))

    return [X_train, y_train, X_test, y_test]
```

- **The seq_len** parameter determines how far back the model will look at historical data, helping the model to capture time dependencies in a memory-aware way.
- We should note that if "seq_len" is too large, the model can become complex and prone to overlearning.
- We can specify separate seq_len values for RNN and LSTM
- I Decided Seq_len = 20, because high seq-len causes overfitting
-

```
seq_len = 20

# Let's create train, test data
X_train, y_train, X_test, y_test = load_data(df, seq_len)

print('X_train.shape = ',X_train.shape)
print('y_train.shape = ', y_train.shape)
print('X_test.shape = ', X_test.shape)
print('y_test.shape = ',y_test.shape)
```

```
X_train.shape =  (110000, 20, 1)
y_train.shape =  (110000,)
X_test.shape =  (6169, 20, 1)
y_test.shape =  (6169,)
```

-

3. Model Development and Configuration

- Build RNN, LSTM model with different activation functions Relu, Tanh, Elu,

RNN model

Layer (type)	Output Shape	Param #
simple_rnn_3 (SimpleRNN)	(None, 20, 40)	1680
dropout_3 (Dropout)	(None, 20, 40)	0
simple_rnn_4 (SimpleRNN)	(None, 20, 40)	3240
dropout_4 (Dropout)	(None, 20, 40)	0
simple_rnn_5 (SimpleRNN)	(None, 40)	3240
dropout_5 (Dropout)	(None, 40)	0
dense_1 (Dense)	(None, 1)	41

Total params: 8201 (32.04 KB)
Trainable params: 8201 (32.04 KB)
Non-trainable params: 0 (0.00 Byte)

```
rnn_model.compile(optimizer="adam",loss="MSE")
rnn_model.fit(X_train, y_train, epochs=10, batch_size=1000)
```

```
rnn_predictions = rnn_model.predict(X_test)

rnn_score = r2_score(y_test,rnn_predictions)
print("R2 Score of RNN model = ",rnn_score)
```

LSTM model

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 20, 40)	6720
dropout_6 (Dropout)	(None, 20, 40)	0
lstm_1 (LSTM)	(None, 20, 40)	12960
dropout_7 (Dropout)	(None, 20, 40)	0
lstm_2 (LSTM)	(None, 40)	12960
dropout_8 (Dropout)	(None, 40)	0
dense_2 (Dense)	(None, 1)	41

Total params: 32681 (127.66 KB)
Trainable params: 32681 (127.66 KB)
Non-trainable params: 0 (0.00 Byte)

```
lstm_model.compile(optimizer="adam",loss="MSE")
lstm_model.fit(X_train, y_train, epochs=10, batch_size=1000)
```

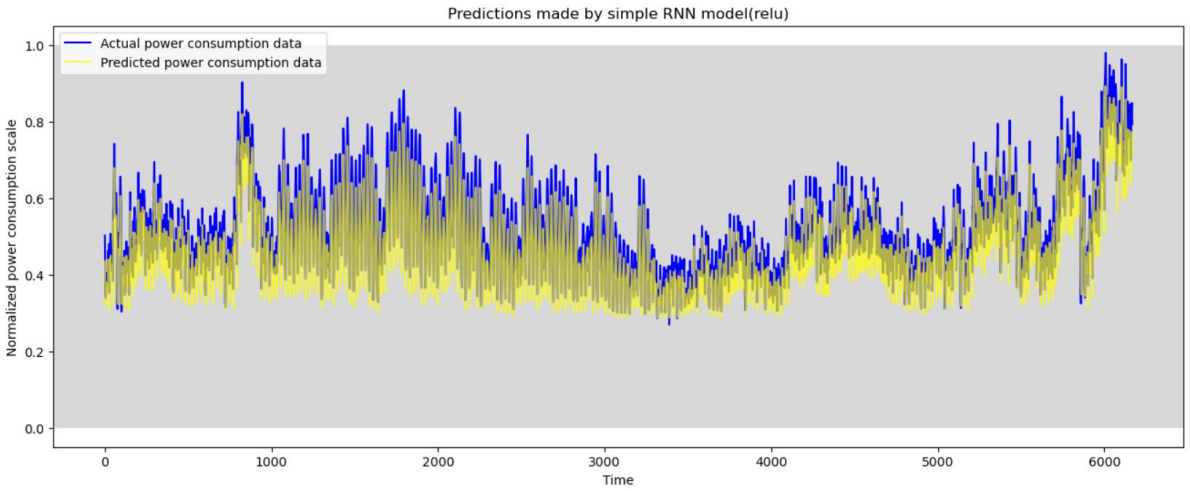
```
lstm_predictions = lstm_model.predict(X_test)

lstm_score = r2_score(y_test, lstm_predictions)
print("R^2 Score of LSTM model = ",lstm_score)
```

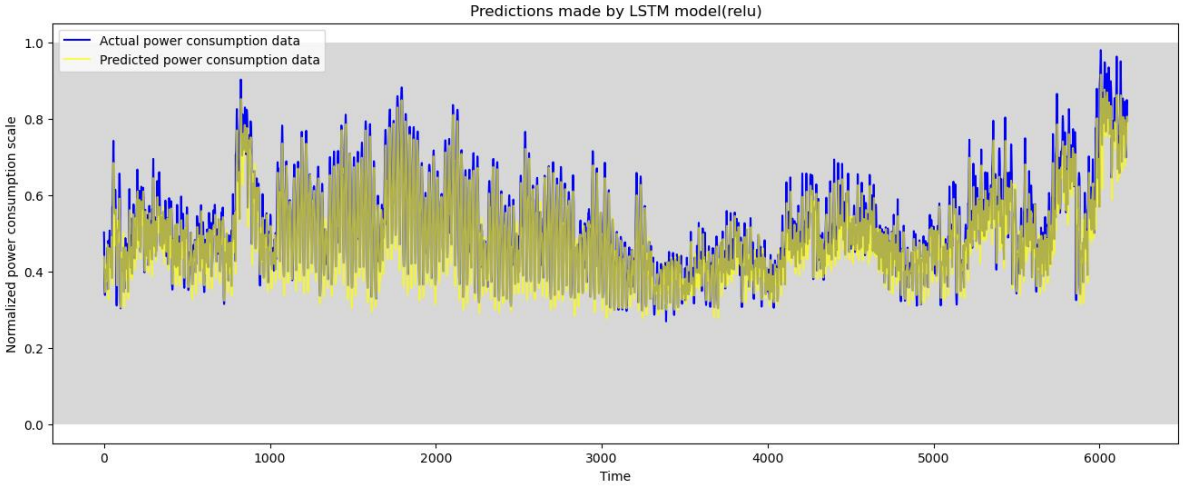
4. Results and Performance Evaluation

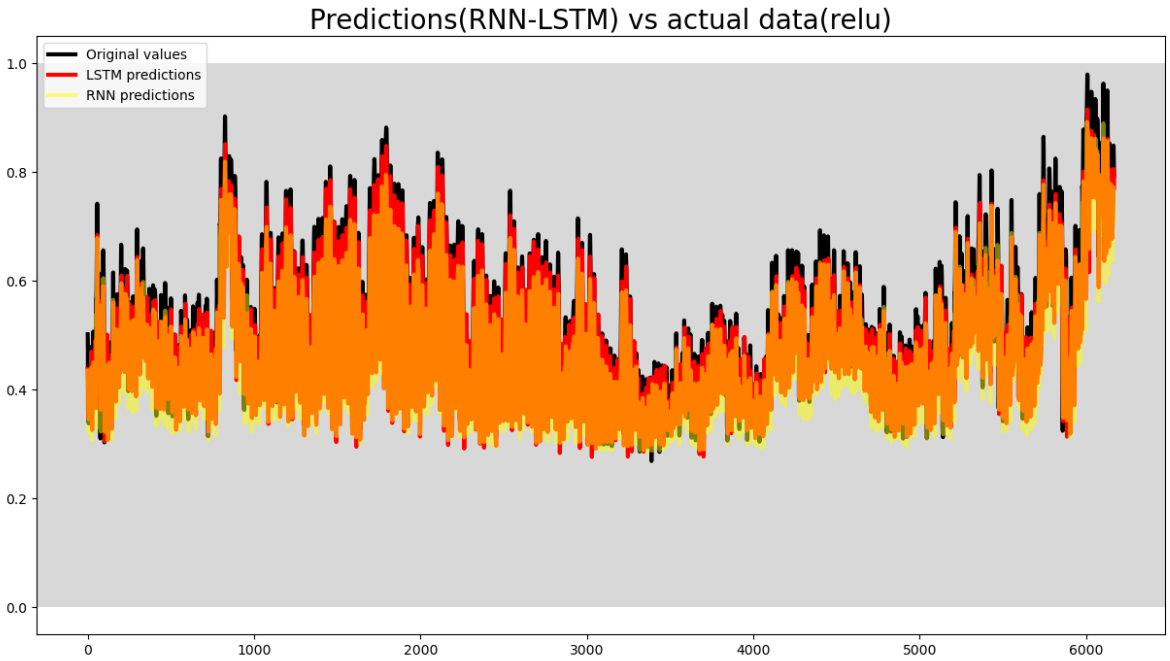
Activation Function	optimizer="adam",loss="MSE" , epochs=10, batch_size=1000, seq_len = 20			
	RNN-R^2	RNN Parameters	LSTM-R^2	LSTM Parameters
Relu	0.7733229	8201	0.8768467	32681
Tanh	0.9472170	8201	0.9522458	32681
Elu	0.9349027	8201	0.9538630	32681

❖ RELU

Here, the RNN and LSTM results obtained by using the activation function relu are shown with graphs, while the prediction values and actual values are given together for comparison.



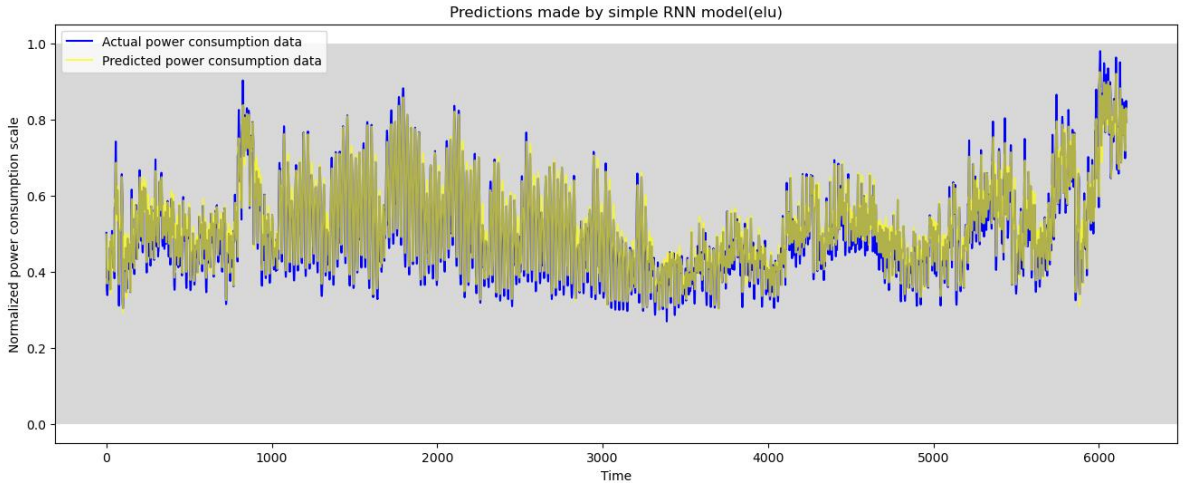


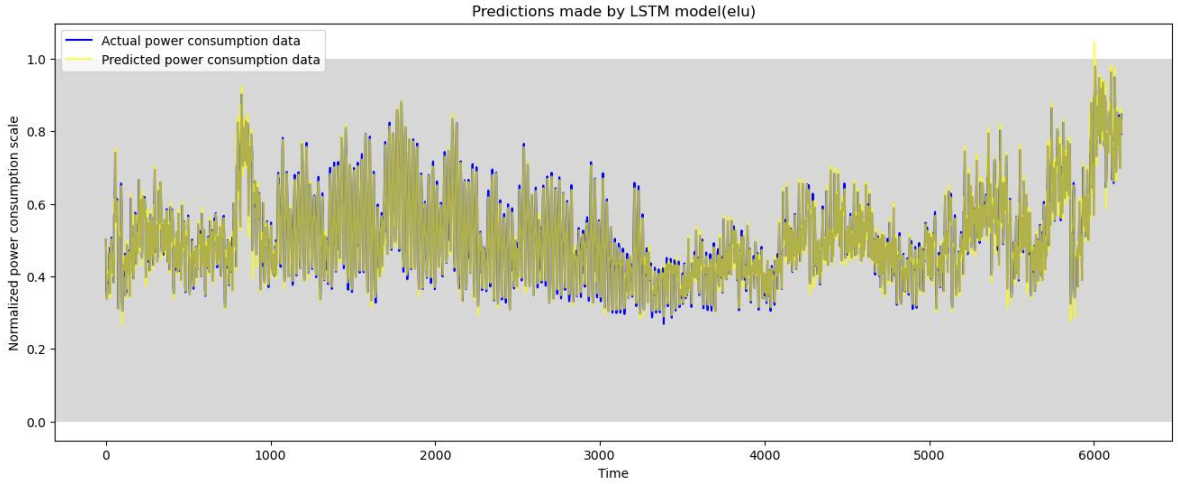


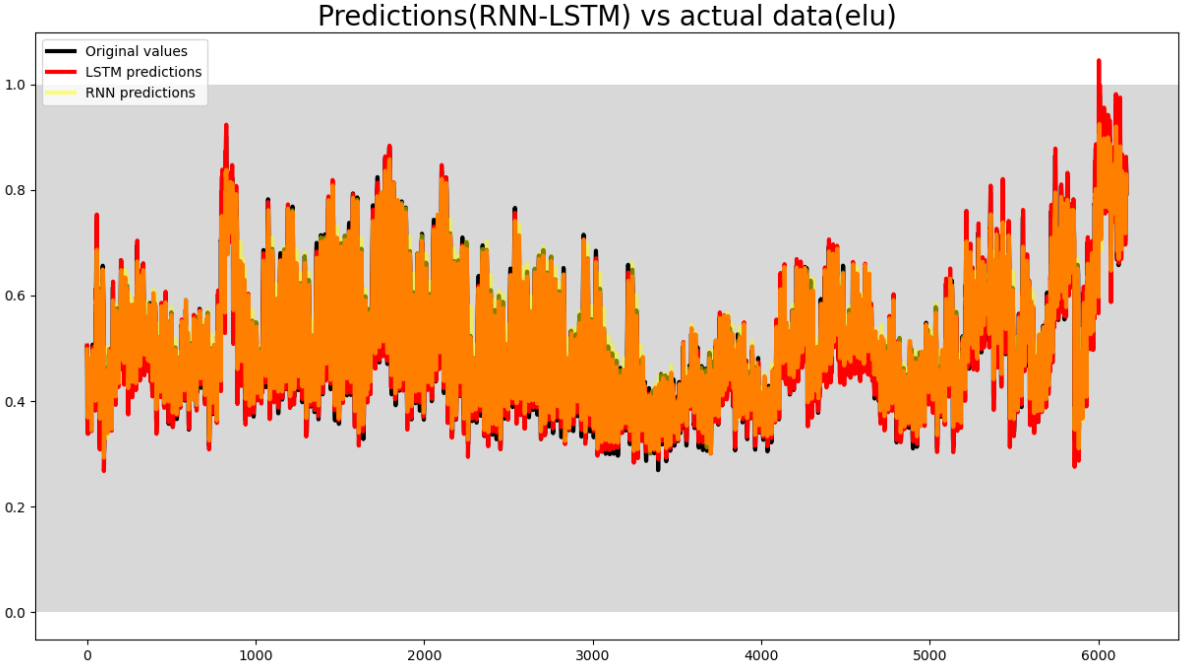
	Date	Actual	RNN Prediction	LSTM Prediction
0	2017-02-13 23:00:00	11494.0	10134.442383	10196.165039
1	2017-02-14 00:00:00	10975.0	9579.812500	10043.628906
2	2017-02-12 01:00:00	8728.0	9173.748047	9627.000977
3	2017-02-12 02:00:00	8390.0	8442.432617	9075.704102
4	2017-02-12 03:00:00	8283.0	7987.369141	8835.267578
...	...	...	...	...
6164	2018-01-01 20:00:00	18418.0	16996.738281	16940.917969
6165	2018-01-01 21:00:00	18567.0	16876.984375	17171.841797
6166	2018-01-01 22:00:00	18307.0	16749.015625	17688.966797
6167	2018-01-01 23:00:00	17814.0	16163.825195	17586.376953
6168	2018-01-02 00:00:00	17428.0	15699.504883	16982.302734

❖ ELU

Here, the RNN and LSTM results obtained by using the activation function elu are shown with graphs, while the prediction values and actual values are given together for comparison.



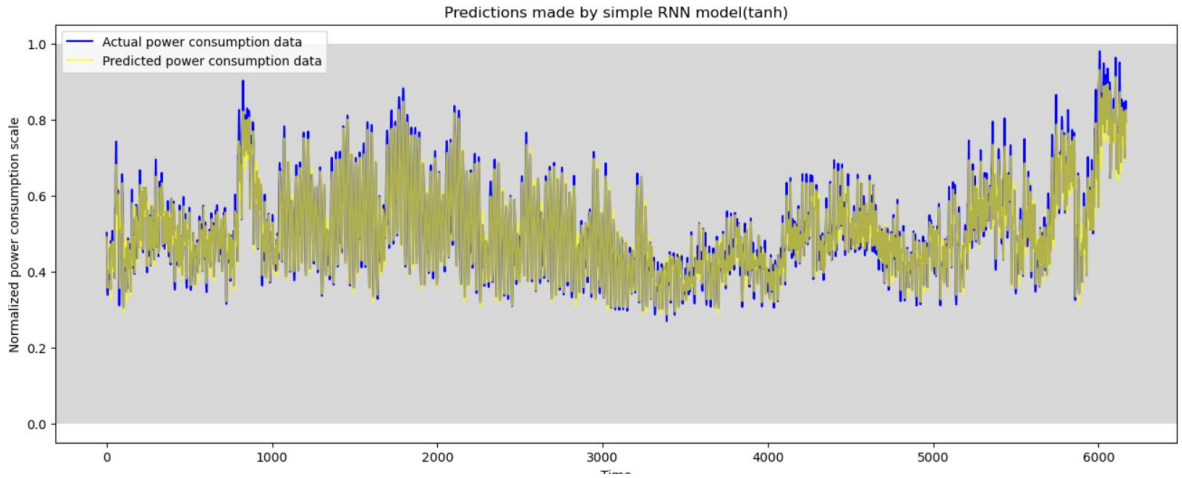


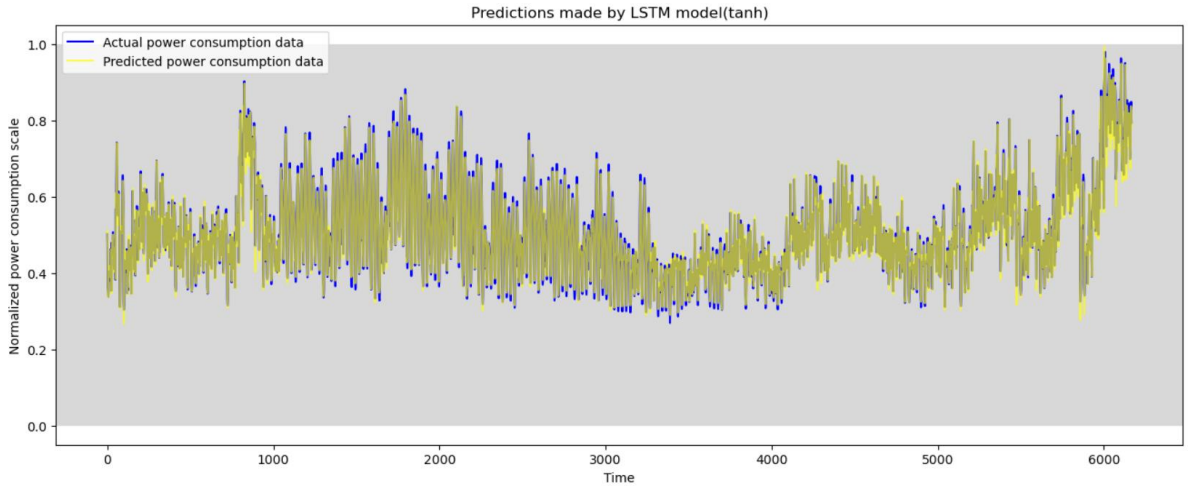


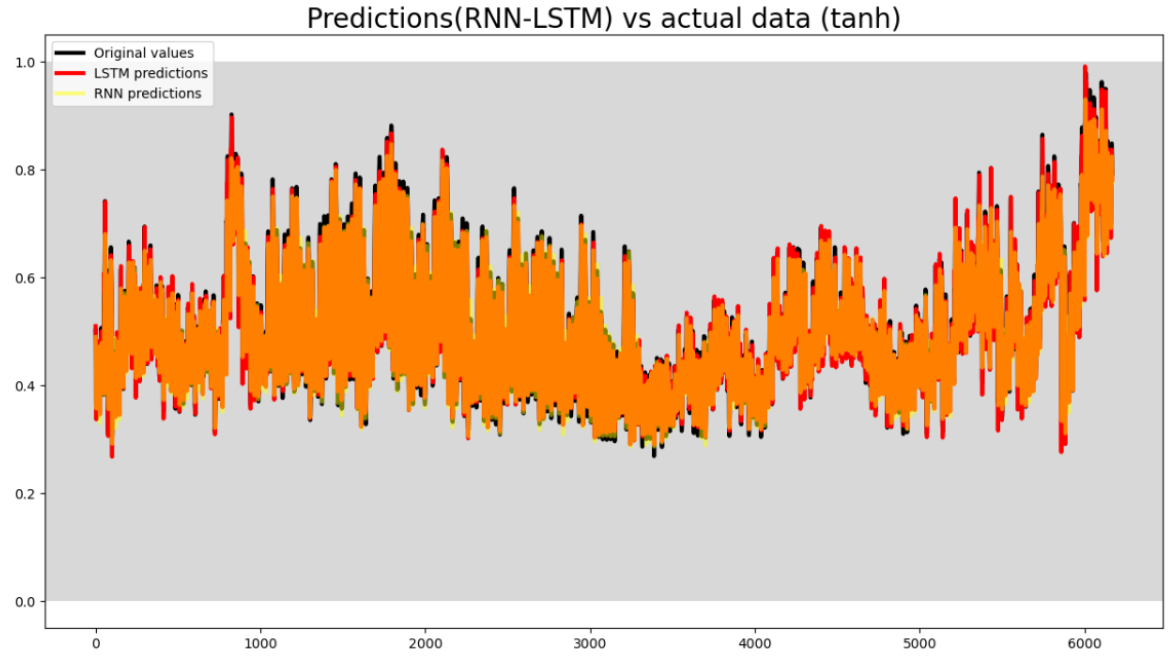
	Date	Actual	RNN Prediction	LSTM Prediction
0	2017-02-13 23:00:00	11494.0	11407.116211	11559.167969
1	2017-02-14 00:00:00	10975.0	10790.358398	10957.188477
2	2017-02-12 01:00:00	8728.0	10476.704102	10537.632812
3	2017-02-12 02:00:00	8390.0	9514.398438	8900.321289
4	2017-02-12 03:00:00	8283.0	8767.454102	8145.400391
...	...	...	...	...
6164	2018-01-01 20:00:00	18418.0	18177.472656	18849.701172
6165	2018-01-01 21:00:00	18567.0	18141.263672	18347.277344
6166	2018-01-01 22:00:00	18307.0	18125.757812	18258.468750
6167	2018-01-01 23:00:00	17814.0	17935.722656	18009.732422
6168	2018-01-02 00:00:00	17428.0	17431.820312	17484.052734

❖ TANH

Here, the RNN and LSTM results obtained by using the activation function tanh are shown with graphs, while the prediction values and actual values are given together for comparison.




Predictions made by LSTM model(tanh)
Actual power consumption data
Predicted power consumption data
Normalized power consumption scale
Time
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
6000


Predictions(RNN-LSTM) vs actual data (tanh)
Original values
LSTM predictions
RNN predictions
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
6000

	Date	Actual	RNN Prediction	LSTM Prediction
0	2017-02-13 23:00:00	11494.0	11248.514648	11658.520508
1	2017-02-14 00:00:00	10975.0	10738.583984	11089.238281
2	2017-02-12 01:00:00	8728.0	10286.926758	10633.527344
3	2017-02-12 02:00:00	8390.0	8922.618164	8750.880859
4	2017-02-12 03:00:00	8283.0	8544.047852	8132.517090
...	...	...	...	...
6164	2018-01-01 20:00:00	18418.0	17839.269531	18329.218750
6165	2018-01-01 21:00:00	18567.0	18004.425781	18071.855469
6166	2018-01-01 22:00:00	18307.0	18079.367188	18086.357422
6167	2018-01-01 23:00:00	17814.0	17711.484375	17808.031250
6168	2018-01-02 00:00:00	17428.0	17093.890625	17194.578125

When we compared the RNN and LSTM models built in the same architecture with different activation functions at the same parameters, I observed that the LSTM model made better predictions than RNN. Again, while the r2 score gave the highest score with the tanh function in the RNN model, similar results were obtained with tanh and elu functions in the LSTM model. Relu activation function lagged behind the others in both models.

5. Conclusions

Different activation functions were used in the RNN and LSTM models, which were built in the same architecture and organized with the same parameters. the reasons why the tanh function gives better results than relu and elu may be as follows.

- Activation Functions and Performance Differences in Electricity Distribution Company Data.
- Activation functions are critical components that affect the performance of neural networks.
- The preference for the tanh activation function in RNN and LSTM models tends to better preserve past information, especially in LSTM layers .
- The tanh function produces an output that is more symmetric with respect to negative and positive inputs, whereas ReLU and ELU can produce zero output by focusing on positive inputs. This can especially help to better model the variations of energy data over time.
- The advantage of Tanh is that it shows a more sensitive response at values close to zero, which is especially useful for time series data.
- ReLU and ELU generally perform better on large data sets and deep neural networks, but special cases in energy data may cause tanh to be more effective .