Deep Learning Project Report

Part – 1

Household Electric Power Demand Forecasting

I. Project Objective

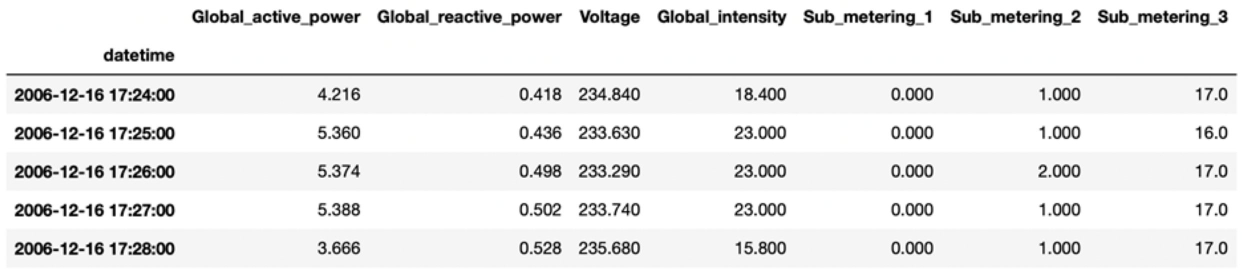
The objective of this report is to analyze the past 30 days daily power requirement of a household and use RNN and LSTM based deep learning models to estimate the expected hourly power usage for the next day, which will provide valuable insights for power distribution companies.

II. Introduction

The Household Power Consumption dataset contains detailed electricity consumption data from a single household in France between December 2006 and November 2010. The dataset contains 2,075,259 observations in per minute frequency, and 8 features, including the timestamp which allows for analysis of how the household's electricity consumption patterns change over time. Out of the eight features, two were used to engineer the target feature of total household power consumption. This derived feature was then used to predict household electric power consumption for a day based on past 30 days' consumption.

III. Data Loading and Cleaning

Below is the head of the dataset loaded into pandas-



	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
datetime							
2006-12-16 17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
2006-12-16 17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2006-12-16 17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
2006-12-16 17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
2006-12-16 17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0

As part of data cleaning, column names were cleaned by converting them to lowercase. Furthermore, the '?' in the dataset was replaced with NaN (Not a Number) and all dtypes were converted to float.

Creating Target Feature

As per the objective, a target feature of total household power consumption, or 'Global Apparent Power', was created using two features in the dataset as-

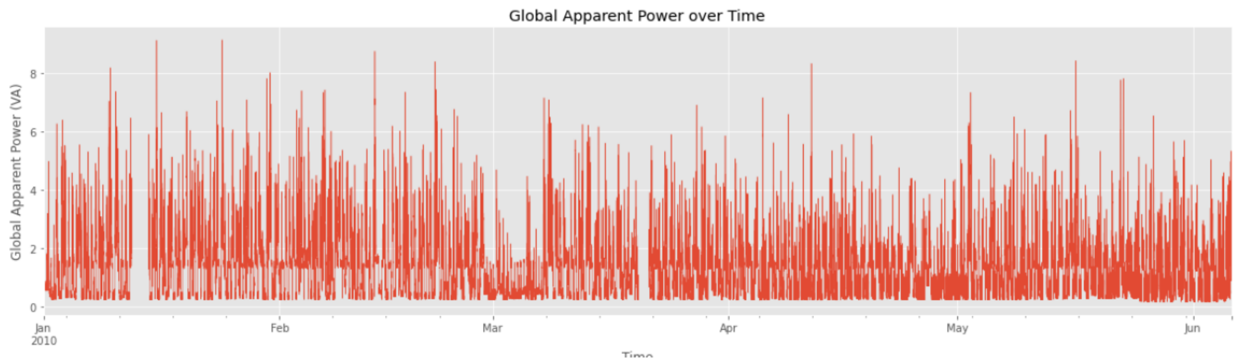
$$Global\_Apparent\_Power = (Global\_Active\_Power^2 + Global\_Reactive\_Power^2)^{½}$$

Features other that global apparent power were now irrelevant to the analysis and were dropped.

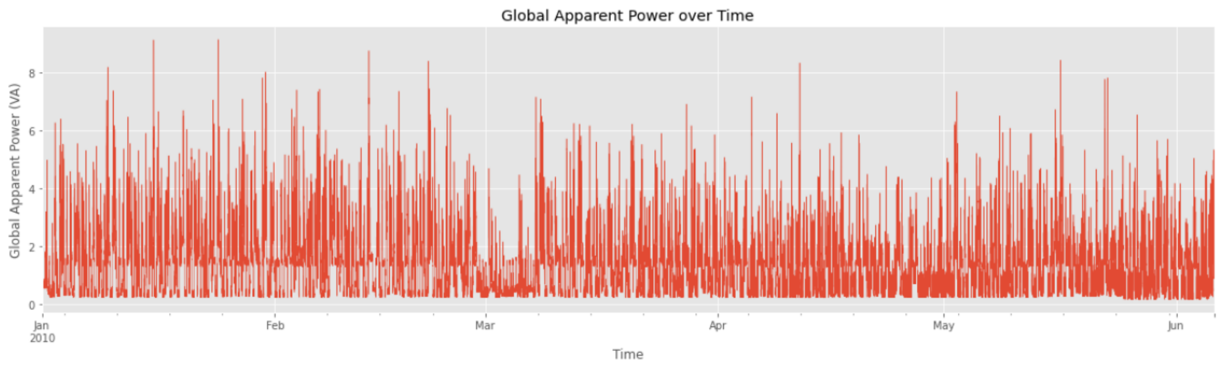
Null Values

Around 25000 records were missing in the dataset. They were forward filled with values from the same day and time from the past week. This was done to preserve cyclic trends in the data.

Data with Null Values:

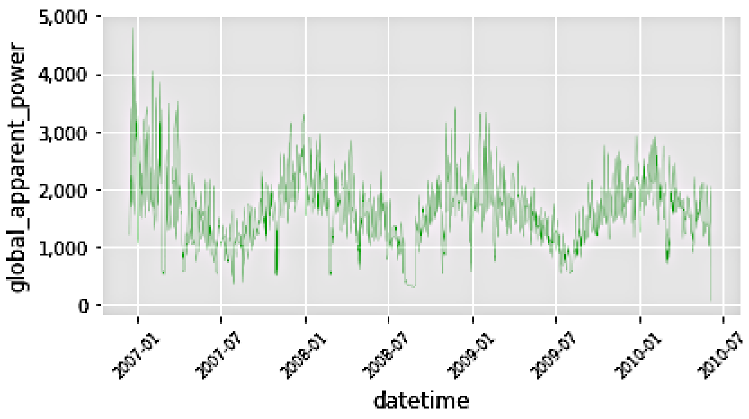


Data with Forward-filled Null Values:



IV. Data Transformation and EDA

The data which was available in per minute frequency, was aggregated to daily values to serve the objective. This revealed a cyclic pattern in the data with an apparent cycle of one year, where power consumption crests at the start and end of the year, and troughs at around August, for each of the 4 years.

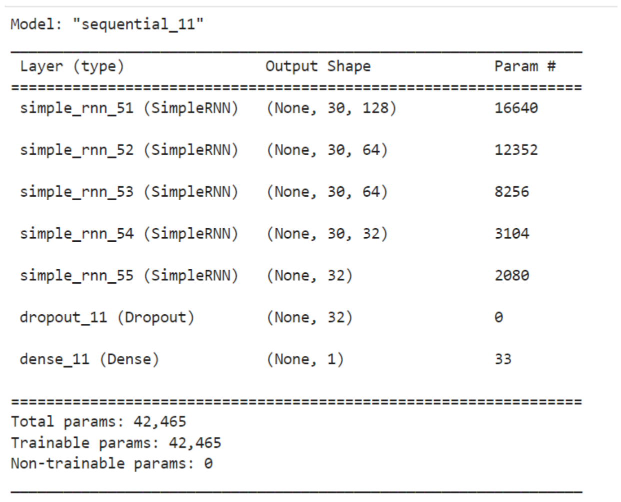


Therefore, it was deemed that both RNN and LSTM models would be suitable for this purpose due to their ability to retain information from previous time steps.

V. Deep Learning Models

The loss function selected for this purpose is mean squared error. Four different models were built to perform this task, and each model performed fairly well. All architectures conserved a basic pyramidal structure with decreasing number of nodes in deeper layers. Each model utilized 'Adam' optimizer for training, and was trained for 100 epochs with conditional callback to save only the epoch version with least validation loss. All dropout parameters were set to 0.2. For models utilizing RNN layers, learning rate parameter was set to 0.0001.

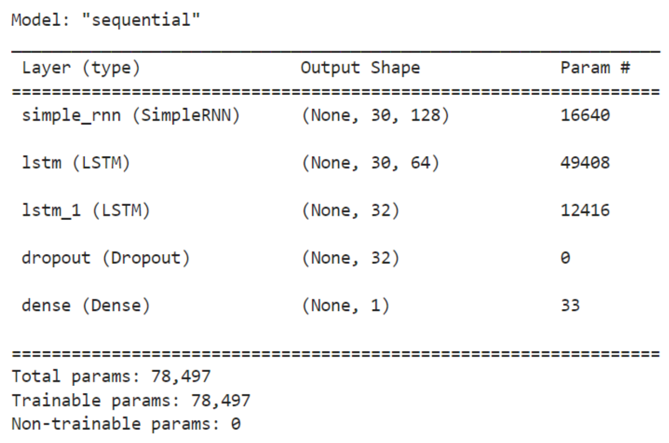
1. Simple RNN model



```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_51 (SimpleRNN)   (None, 30, 128)           16640

 simple_rnn_52 (SimpleRNN)   (None, 30, 64)            12352

 simple_rnn_53 (SimpleRNN)   (None, 30, 64)            8256

 simple_rnn_54 (SimpleRNN)   (None, 30, 32)            3104

 simple_rnn_55 (SimpleRNN)   (None, 32)                2080

 dropout_11 (Dropout)        (None, 32)                0

 dense_11 (Dense)            (None, 1)                 33

=================================================================
Total params: 42,465
Trainable params: 42,465
Non-trainable params: 0
_________________________________________________________________
```

This model utilized tanh activation in all RNN layers, and relu activation for the dense layer.

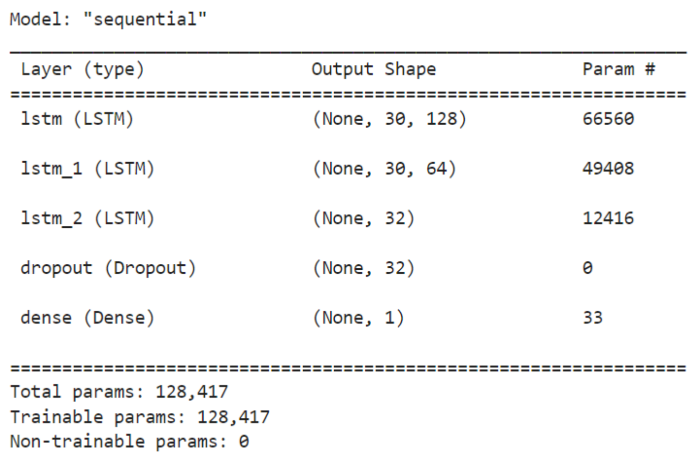
2. RNN-LSTM model



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 30, 128)           16640

 lstm (LSTM)                 (None, 30, 64)            49408

 lstm_1 (LSTM)               (None, 32)                12416

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 78,497
Trainable params: 78,497
Non-trainable params: 0
```

The first layer for this model was kept as a simple RNN layer, after which two LSTM layers were added. Tanh activation was used in all layers except the final dense layer, which used no activation.

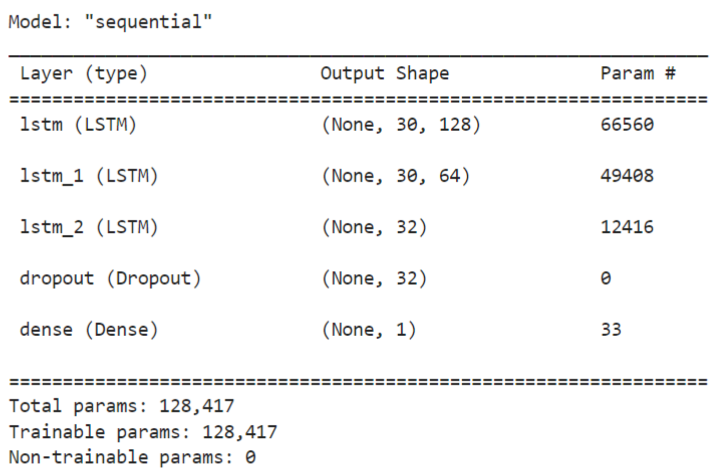
3. LSTM model with relu activation



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 128)           66560

 lstm_1 (LSTM)               (None, 30, 64)            49408

 lstm_2 (LSTM)               (None, 32)                12416

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 128,417
Trainable params: 128,417
Non-trainable params: 0
```

Three LSTM layers, each with relu activation, and a dense layer with no activation.

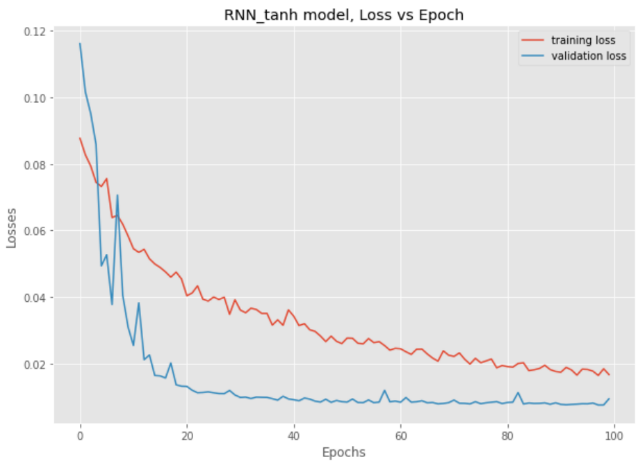
4. LSTM model with tanh activation

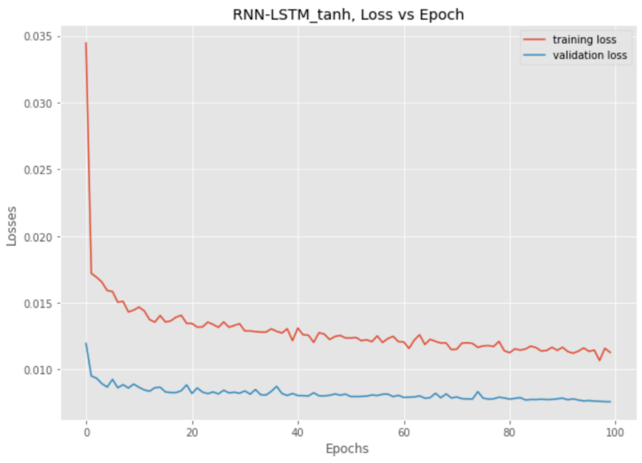


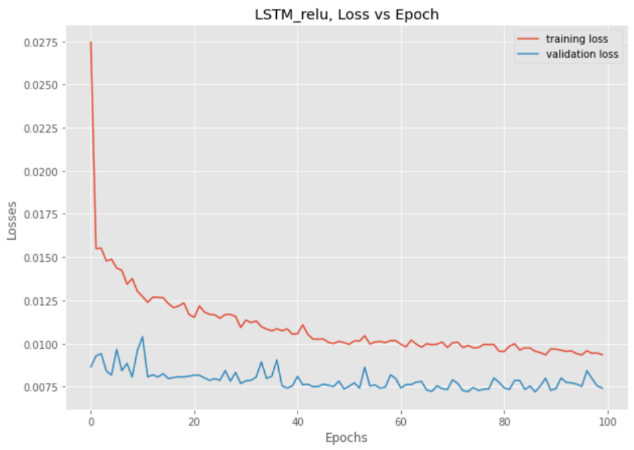
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 128)           66560

 lstm_1 (LSTM)               (None, 30, 64)            49408

 lstm_2 (LSTM)               (None, 32)                12416

 dropout (Dropout)           (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 128,417
Trainable params: 128,417
Non-trainable params: 0
```

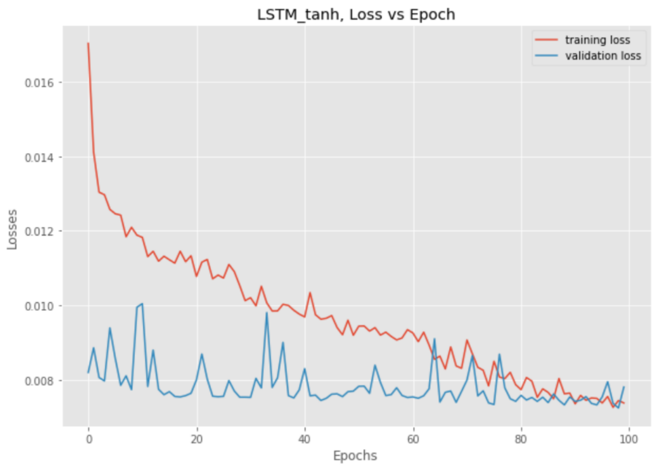
Three LSTM layers, each with tanh activation, and a dense layer with no activation.

Learning Curves









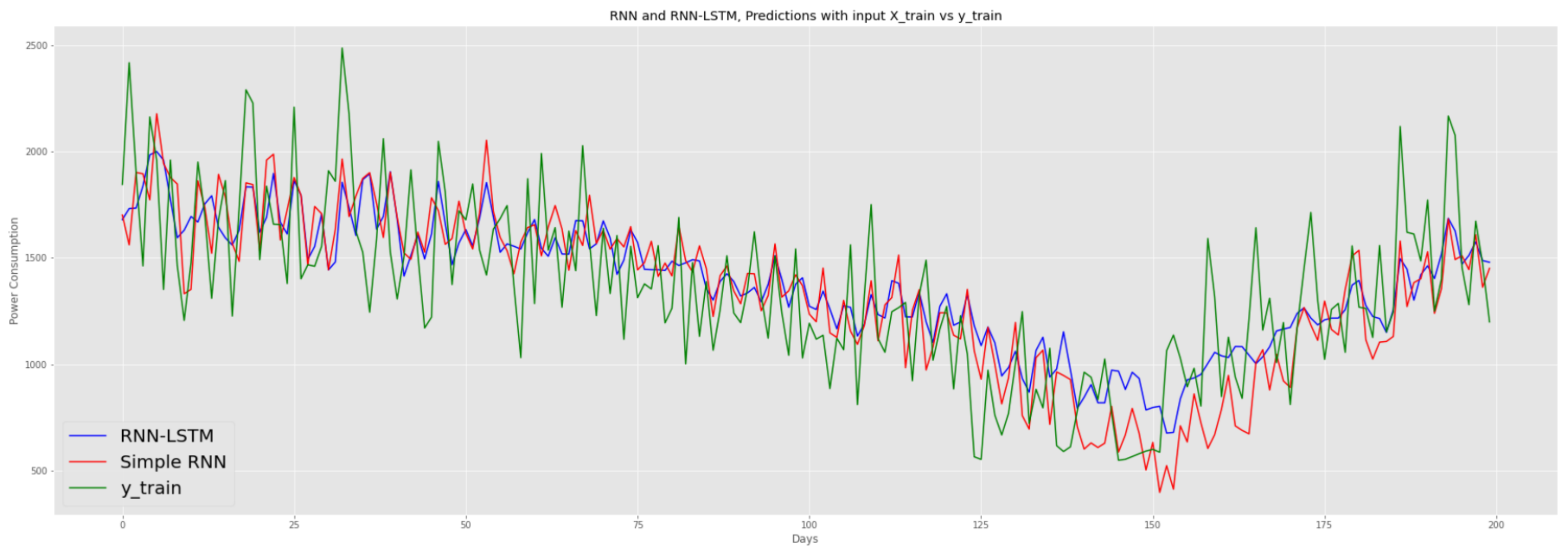
All models show decrease in training loss with increasing epochs. However, models using RNN layers can be seen as reaching optimum epoch with a distinct elbow in validation loss, which is not the case with the LSTM models.

This can be because of LSTM's ability to better retain previous information, which in this dataset, might be noise.

Predictions on Training and Test Data

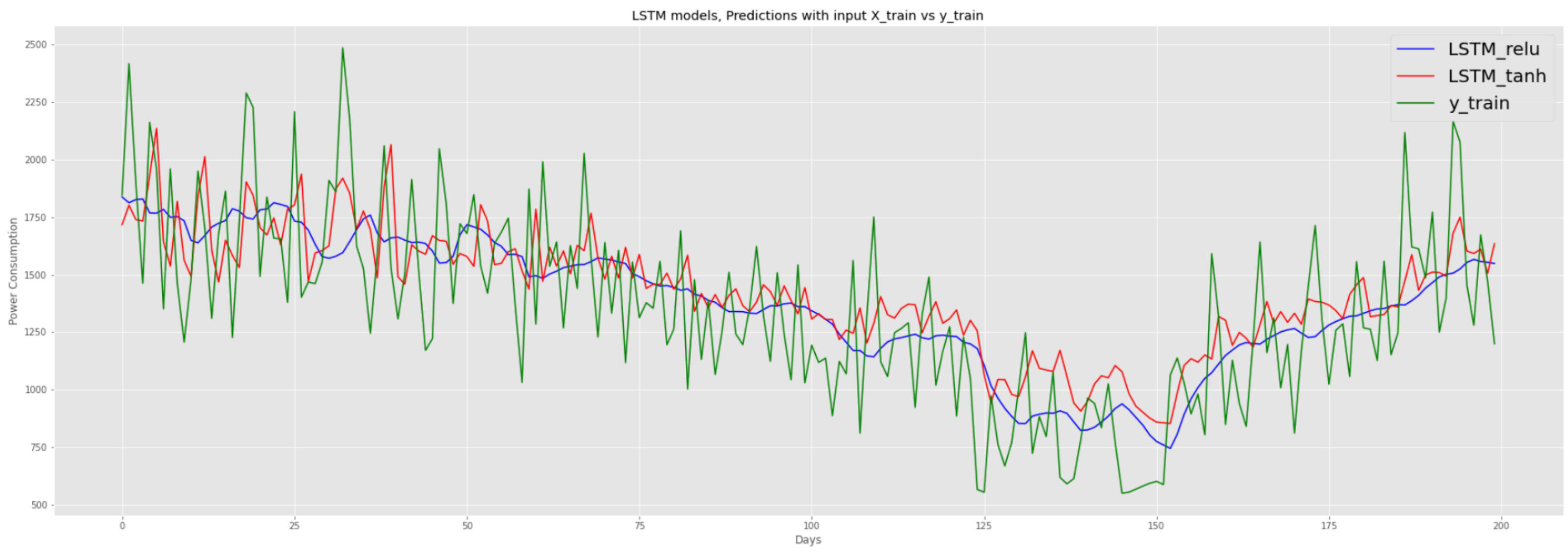
Last 200 days and last 100 days of data and predictions have been used to visualize training and test predictions respectively.

<u>RNN and RNN-LSTM- Training Predictions</u>



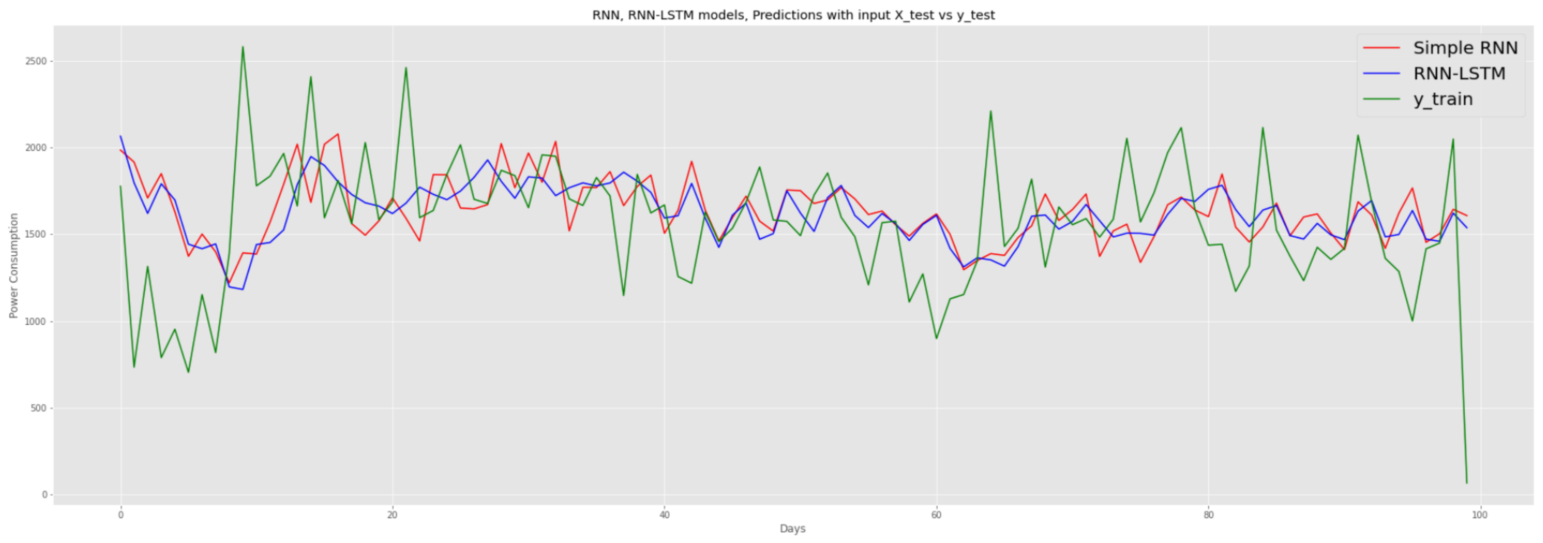
Both models seem to have performed equally good, however RNN-LSTM model tends to follow the general trend of the data, whereas Simple RNN seems to simulate the fluctuations in data.

<u>LSTM-relu and LSTM-tanh- Training Predictions</u>

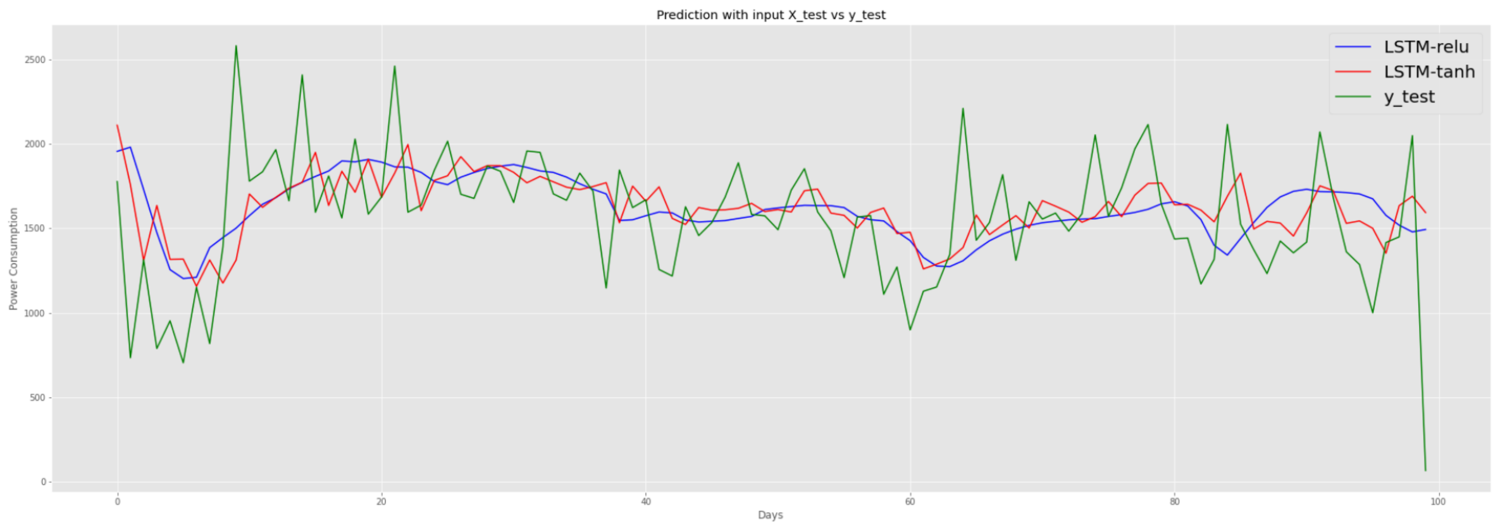


The LSTM-relu model excellently captures the trend, however fails to capture the intricacies of the variations in the data, which might be due to the linearity of relu activation. The LSTM-tanh model seems to have added variance when compared to the LSTM-relu model.

<u>RNN and RNN-LSTM- Test Predictions</u>



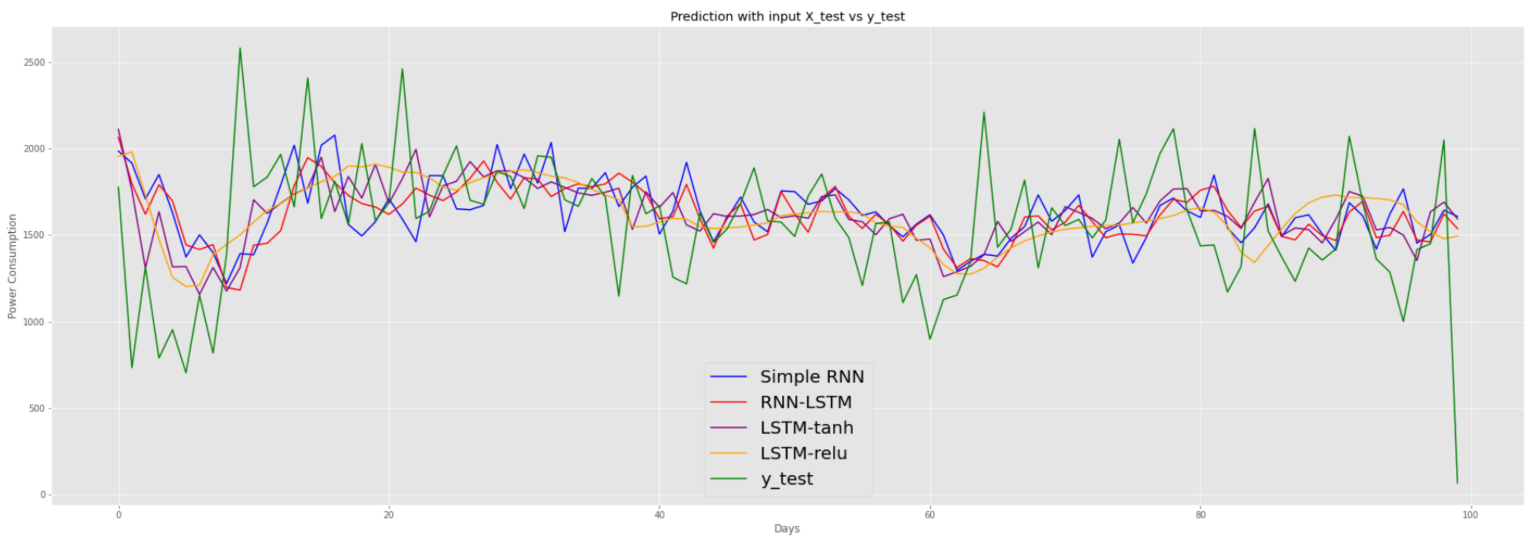
<u>LSTM-relu and LSTM-tanh- Test Predictions</u>



Similar prediction behavior was observed for test and train data by each model.

VI. Summary & Discussion

All Models- Test Predictions



All models have results in the close vicinity of each other, and the target values. However, some models took a few more minutes to train than the others, and also had an additional number of parameters to be trained.

Summary of Model Comparisons

	Training Time (min:sec)	**Trainable Params**	**Val loss (MSE)**
Simple RNN	10:48	42465	0.0076
RNN-LSTM	11:42	78497	0.0074
LSTM tanh	12:50	128417	0.0073
LSTM relu	14:35	128417	0.0072

Based on the loss function, one might be tempted to choose the LSTM-relu model. However, the LSTM-relu model is a low variance and high bias model, which will adequately follow the trend, but will also give conservative results in terms of sudden fluctuations in data, which is an important consideration when forecasting electric power demand. Therefore, among the models in consideration, the best model for this case would be the **LSTM-tanh model**, which seems to provide an appropriate balance of model variance and bias.

Part – 2
HoverAce – Game Development using Reinforcement Learning

I. Introduction:

In recent years, deep reinforcement learning has shown great promise in training agents for complex tasks that require decision making and strategy, such as playing games, controlling robots, and driving cars. In this report, we explore the application of deep reinforcement learning to train an AI agent for a futuristic racing game set in a virtual reality environment.

II. Game Description:

The game is a futuristic racing game where players control high-speed hovercrafts and compete in races on a variety of challenging tracks with twists, turns, and attacks. The goal is to finish the race in the shortest time possible while using strategic boosts to gain advantage and deploying timely attacks to hinder opponents. The hovercrafts can move in multiple directions - forward, backward, left, and right. Boosters, shields, and weapons can be collected from checkpoints within the race, which can then be activated for temporary bursts of speed, or for protection against attacks, or use offensive weapons to hinder opponents.

III. Actions:

The actions available to the hovercrafts include moving in multiple directions - forward, backward, left, and right. They can also activate boosters for temporary bursts of speed, deploy shields for protection against collisions, or use offensive weapons to hinder opponents.
Set of actions- {front motion, back motion, left motion, right motion, activate weapon, activate shield, activate booster}
The ideal final activation function for our deep learning model that trains AI players- **Softmax with 7 outputs**, each output representing probability for one action.

IV. States:

The game states include the current position and velocity of the hovercraft, the positions of other racers, the availability of boosts and weapons, and the time elapsed in the race.
Set of states- {Self Position on track, self rank on track, speed, direction, opponent position on track, opponent rank on track, available boost, available weapons, time elapsed}

V. Rewards/Penalties:

Reward is an *internal coin system*, which will increase, or decrease for penalty, based on change of certain states as follows-

Self rank improves (overtaking)- *Reward*
Speed increase in correct direction- *Reward*
Collecting boost and shield - *Reward*
Hitting opponent by your attack- *Reward*
Defending opponent's attack by shield- *Reward*

Self goes in wrong direction - *Penalty*
Self rank goes down (being overtaken) - *Penalty*
Self gets hit by an attack - *Penalty*
Self goes in wrong direction - *Penalty*

VI. **Approach**:

We propose a deep reinforcement learning approach with a combination of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to train the AI agent. The CNNs would process visual information from the game environment, such as the positions of hovercrafts, track layout, and checkpoints. The RNNs would capture the sequential nature of the game, such as the movements of the hovercraft and the passage of time.

For the state and reward distribution, we would use the states and rewards mentioned above. Based on the present state distribution, the agent would choose an action (or multiple actions, as suggested by the model) from the set of actions, which would result in a change of state. This change of state would be assessed by the reward system, which would accordingly assign a reward or penalty.

VII. **Final Reward:**

The final reward is based on the final rank the agent will achieve, and how soon it finishes the race. All other factors which contribute to the training's internal coin-based reward system are designed to help achieve this final reward only.
However, if in some case, the agent needs to miss out on collecting a checkpoint (which offers coin reward) in order to finish the race with a better rank (final reward), the checkpoint will be rightly missed to prioritize the final reward.

VIII. **Model Training:**

The AI agents would be trained on model-based reinforcement learning techniques. **We use the current state as input features to determine the combination of actions which would maximize the rewards earned during the state change by implementing those actions.** To generate AI agents with different performance efficiency, we can make slight changes to the base model like, using different activation functions and dropouts rates.

The trained neural network would then be used to simulate gameplay and collect additional data for model-based training, where the agent learns to predict the future states and rewards of the game. The agent's performance would be evaluated on a validation set of gameplay data, and the training process would be iteratively refined to improve the agent's performance.

Once the agent achieves a satisfactory level of performance, it can be deployed to race autonomously using the learned action rule and reward/state distribution, making strategic decisions on boosts, weapons, and racing tactics in real-time to win the race.

IX. Summary:

The reward and state distribution would include the current game state, the progress in the race, and the performance of the hovercraft as inputs to the neural network. The final reward would be a combination of the race position and time taken to complete the race. The AI agent would be incentivized to finish the race in the highest rank possible, using boosts, and effectively utilizing offensive weapons to hinder opponents.

This approach can lead to a highly competitive and engaging racing game, where players can race against AI opponents with realistic and intelligent behaviors.