Amazon Spot Prices Prediction With Neural Networks

Savio Berdine
Departamento de Informática
UFRPE
Recife, Brazil
savio.berdine@ufrpe.br

***Abstract*—Spot pricing is a way of buying cloud computing services based on the demand and supply of the current market. It's usually more cost efficient than fixed pricing because users can bid on unused resources for a lower price. But predicting prices can be challenging given the quasi-spontaneous nature of the price driven by demand behavior. In this paper, we use two neural networks approaches to predict the spot price of EC2 AWS instances. We use three months' worth of data provided by AWS and refined by us. The results are promising being able to predict with more than 90% accuracy in every experiment. We also discuss the next steps and how to improve the methods presented here.**

Keywords—spot, pricing, prediction, aws, cloud

I. Introduction

Being able to predict cloud resources prices is imperative for the well-being of modern businesses. One of the price behaviors of modern cloud services markets is the spot pricing. Spot pricing is a way of buying cloud computing services based on the demand and supply. In this article we will explore how to use deep learning techniques to forecast spot prices of EC2 AWS instances. The pricing of the EC2 instances varies based on several factors, including the operating system, the region, and the time of usage. This is a regression problem based on a dataset which is composed by a timeseries. We will explore how to preprocess the data for each neural network and demonstrate the results using Root Mean Squared Error and Mean Absolute Percentage Error metrics.

II. The Dataset

A. Collection

We will use AWS CLI commands to gather the data. AWS provides price history for the past 90 days for its EC2 instances. To do that, we need to create an AWS account, go to Identity and Access Management Console and create an administrator user. With that, we have access to the access key id and the secret access key. The next step is to configure the AWS CLI with these parameters.

We have collected spot price history data for three different Linux distributions offered by AWS - Linux/UNIX, Red Hat Enterprise Linux, and SUSE Linux. The data is composed by five parameters: AvailabilityZone, InstanceType, ProductDescription, SpotPrice and Timestamp. To isolate the analysis by product, instance type and availability zone, we collect data from three different products, but all of them in the same availability zone and the same instance. We do that with the following commands:

- aws ec2 describe-spot-price-history --instance-types m1.large --product-description "SUSE Linux" --availability-zone "us-west-1a" --start-time 2022-10-28T00:00:00 --end-time 2023-04-02T00:00:00 > spot-price-history-suse-linux.json
- aws ec2 describe-spot-price-history --instance-types m1.large --product-description "Red Hat Enterprise Linux" --availability-zone "us-west-1a" --start-time 2022-10-28T00:00:00 --end-time 2023-04-02T00:00:00 > spot-price-history-red-hat.json
- aws ec2 describe-spot-price-history --instance-types m1.large --product-description "Linux/UNIX" --availability-zone "us-west-1a" --start-time 2022-10-28T00:00:00 --end-time 2023-04-02T00:00:00 > spot-price-history-Linux-Unix.json

Note that these commands will only work properly when we put the most recent date in the end-time parameter because AWS will only provide data from the last 90 days.

B. Data processing

Initially, we transform the data in na array of price history collections. AWS CLI provides us with a JSON object with key "SpotPriceHistory" containing that forementioned array.

We noted that, in depending on the way we configure the AWS commands, some duplicates in the timestamp can appear, so we drop the duplicates. We also noted that the time intervals in the time series are not constant. Previous works

solve this by calculating the mean price for each day and considering this a problem with constant intervals of one day. Here, we understand that the non-constant intervals are an important characteristic of the behavior of the input data. So, we leave the time series in this format.

Given that, the relevant input for the neural networks are the prices, ordered by timestamps. So we keep only the spot price column. We then normalize the data using the MinMaxScaler class from the sklearn.preprocessing module, given that we will be using the sigmoid activation function on our tests.

III. Model Training and Evaluation

We train three different deep learning models to predict the spot prices of the EC2 instances - a Multi-Layer Perceptron (MLP), a Long Short-Term Memory (LSTM) Network and a Convolutional Neural Network (CNN). The models take a look-back window of n previous spot prices to predict the next price. We split the data into training and testing sets in a 67% - 33% proportion and evaluate the models based on their root mean squared error (RMSE) and mean absolute percentage error (MAPE).

A. Training the LSTM Network

Using the Sequiential API from Keras framework, we create an LSTM network composed by two LSTM layers with 50 and 5 cells respectively and one Dense layer to make a unique prediction. The model is compiled with a mean squared error loss function and an Adam optimizer. The training is performed with a batch size of 16 and for 200 epochs.

The trained model is then used to make predictions on both the training and test datasets. The predictions are then inverted using the scaler used to normalize the data. The root mean squared error (RMSE) and mean absolute percentage error (MAPE) are calculated for both the training and test predictions. Finally, the predictions are plotted along with the original data using python's matplotlib.

B. Training the MLP

Using the Sequiential API from Keras framework, the MLP model is defined with three layers: the first two layers have 50 and 5 nodes respectively and use the rectified linear unit (ReLU) activation function, while the last layer has a single node with no activation function. The model is compiled with the mean squared error loss function and the Adam optimizer.

The model is trained for 200 epochs with a batch size of 16 and a validation split of 0.2. After training, the model is used to make predictions on the training and test sets, and the predicted values are inverted using the scaler.inverse_transform function. The root mean squared error (RMSE) and mean absolute percentage error (MAPE) are calculated for both the training and test sets and presented.

Finally, the predicted values are shifted and plotted against the original data using matplotlib.

C. Training the CNN

The training and testing datasets are reshaped into the expected shape for a 1D CNN model. Then, a Sequential model is created with multiple layers: two 1D convolutional layers with ReLU activation, each followed by a max pooling layer, and then two fully connected layers with ReLU activation and a final output layer with a single neuron. The model is compiled with a mean squared error loss function and an Adam optimizer. The training is performed with a batch size of 16 and for 200 epochs.

As is the case with the previous networks, the trained model is used to make predictions on both the training and test datasets. The predictions are then inverted using the scaler used to renormalize the data. The root mean squared error (RMSE) and mean absolute percentage error (MAPE) are calculated for both the predictions. Finally, the predictions are plotted along with the original data using python's matplotlib.

IV. Results and Discussions

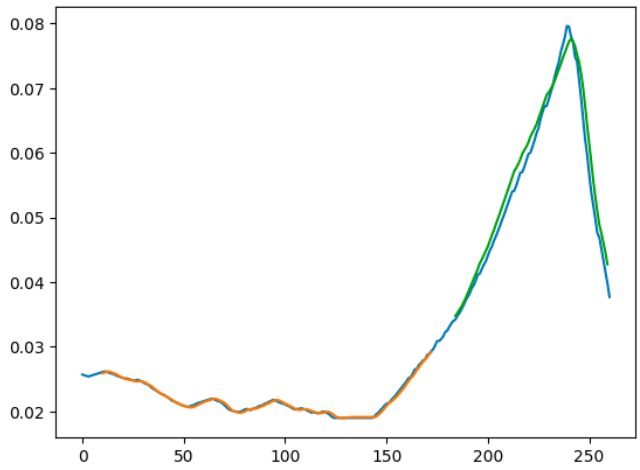
In our tests, the network with the most consistent positive results, always scoring with at most 10% error rate regarding the manipulations on the hyperparameters, was the LSTM. The CNN also performed well, but the MLP performed poorly, often scoring 11% plus error rates.

As we saw in the previous section, our approach was to create networks with similar complexity ergo with similar hyperparameters. The nature of the network guided the small differences in the processing of data, in the expected shape and small differences in hyperparameters. With this approach we intended to compare in a fairer way the results. Nonetheless, is important to note, as we point in the conclusion remarks, that a better fine tuning of the hyperparameters per network can be achieved by various means.

All the graphs bellow have the price as its “y” and the timestep as its “x”.

A. *LSTM results*

First, we make predictions for the Linux/UNIX dataset:

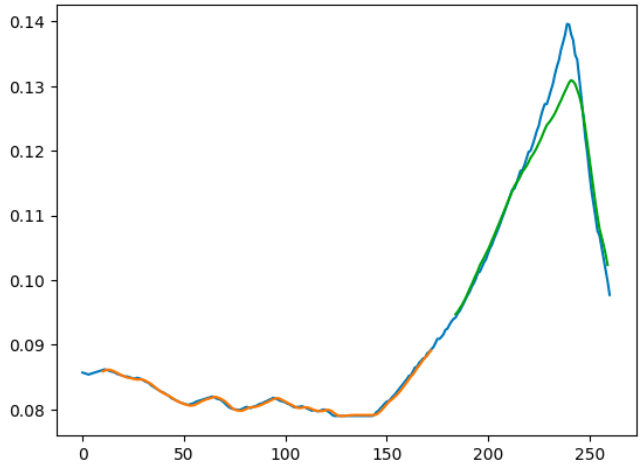


Data in blue, train data prediction in orange, test data prediction in green

TABLE I. LSTM – LINUX/UNIX

	Train	**Test**
RMSE	0.0002	0.0023
MAPE	0.8198%	3.5551%

Secondly, we make predictions for the Red Hat Enterprise Linux dataset:

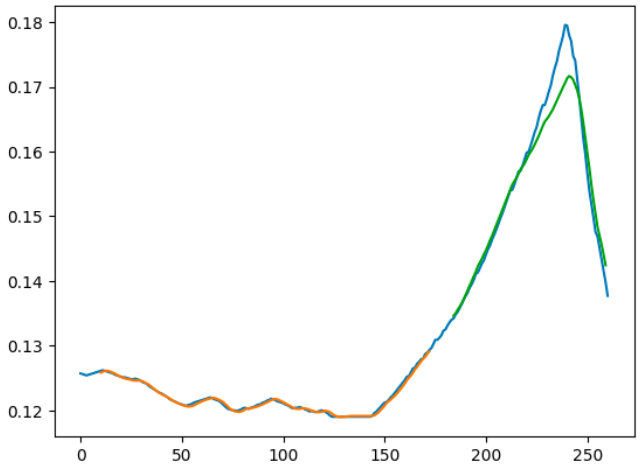


Data in blue, train data prediction in orange, test data prediction in green

TABLE II. LSTM – RED HAT ENTERPRISE LINUX

	Train	**Test**
RMSE	0.0002	0.0033
MAPE	0.2336%	1.7845%

Third, we make predictions for the SUSE Linux dataset:



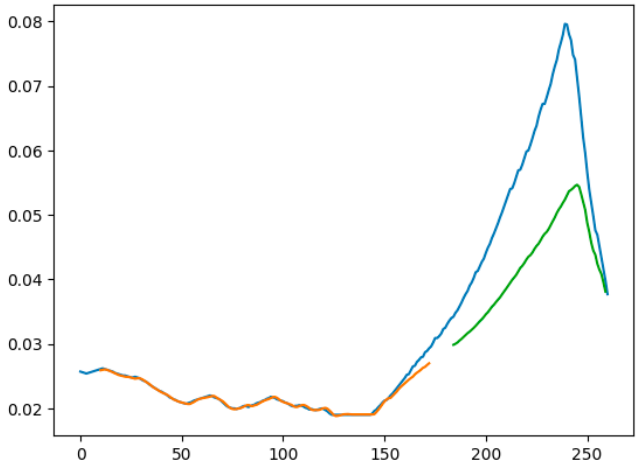
Data in blue, train data prediction in orange, test data prediction in green

TABLE III. LSTM – SUSE LINUX

	Train	**Test**
RMSE	0.0002	0.0030
MAPE	0.1623%	1.2656%

B. *MLP results*

First, we make predictions for the Linux/UNIX dataset:

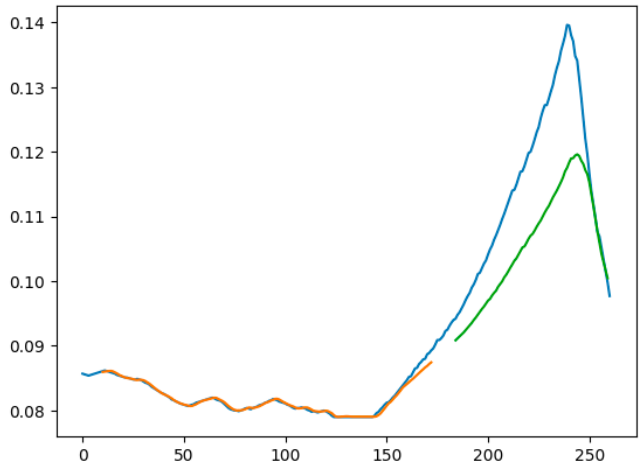


Data in blue, train data prediction in orange, test data prediction in green

TABLE IV. MLP – LINUX/UNIX

	Train	**Test**
RMSE	0.0005	0.0150
MAPE	1.0408%	22.7471%

Secondly, we make predictions for the Red Hat Enterprise Linux dataset:

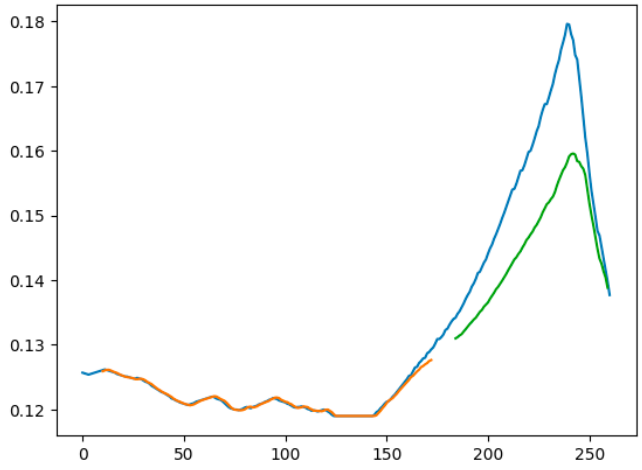


Data in blue, train data prediction in orange, test data prediction in green

TABLE V. MLP – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0004	0.0117
MAPE	0.2782%	8.2360%

Third, we make predictions for the SUSE Linux dataset:



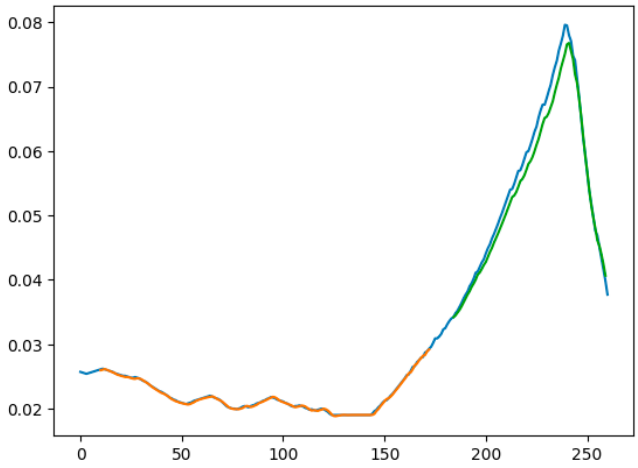
Data in blue, train data prediction in orange, test data prediction in green

TABLE VI. MLP – SUSE LINUX

	Train	Test
RMSE	0.0003	0.0116
MAPE	0.1603%	6.3771%

C. *CNN results*

First, we make predictions for the Linux/UNIX dataset:

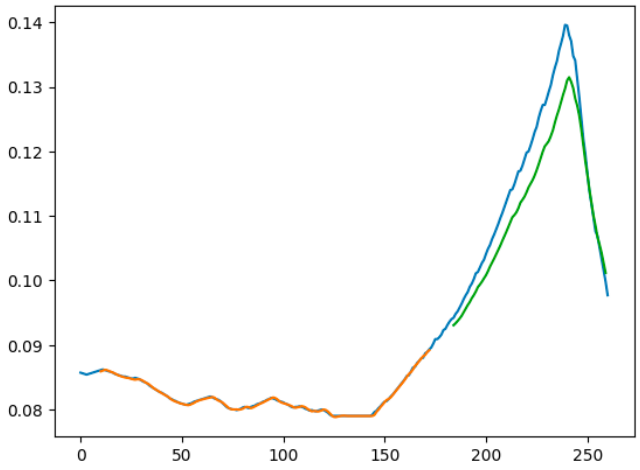


Data in blue, train data prediction in orange, test data prediction in green

TABLE VII. CNN – LINUX/UNIX

	Train	Test
RMSE	0.0001	0.0020
MAPE	0.4730%	2.7823%

Secondly, we make predictions for the Red Hat Enterprise Linux dataset:

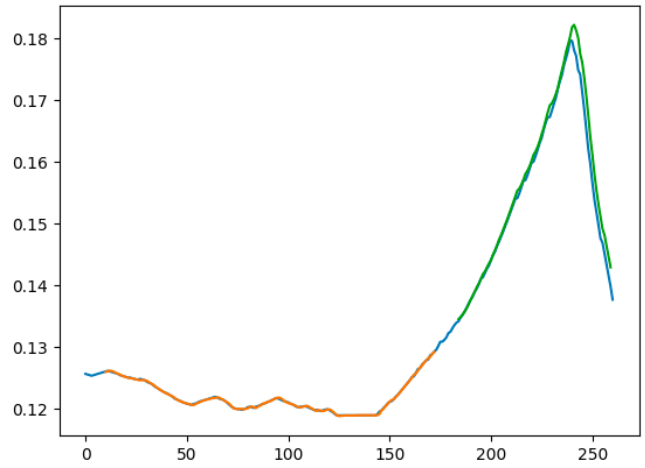


Data in blue, train data prediction in orange, test data prediction in green

TABLE VIII. CNN – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0001	0.0050
MAPE	0.1140%	3.5357%

Third, we make predictions for the SUSE Linux dataset:



Data in blue, train data prediction in orange, test data prediction in green

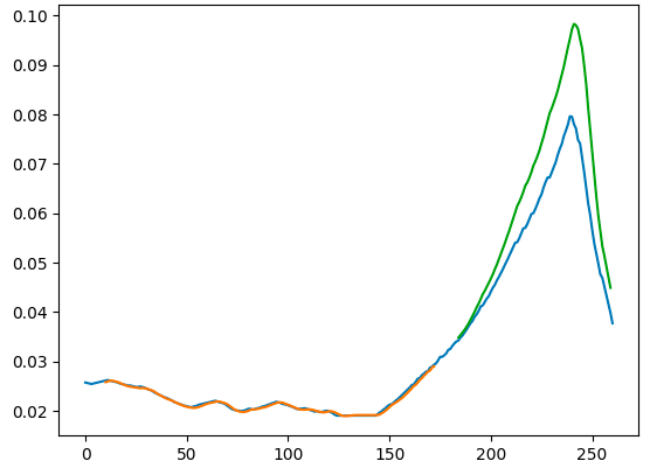
TABLE IX. CNN – SUSE LINUX

	Train	Test
RMSE	0.0001	0.0021
MAPE	0.0633%	0.8498%

D. LSTM results modifing hyperparameters

First, we modify the number of LSTM layers, leaving just one layer with 25 cells plus a Dense layer with one cell to give the regression prediction. We can see by the results bellow that the prediction accuracy diminishes.

We then make predictions for the Linux/UNIX dataset:

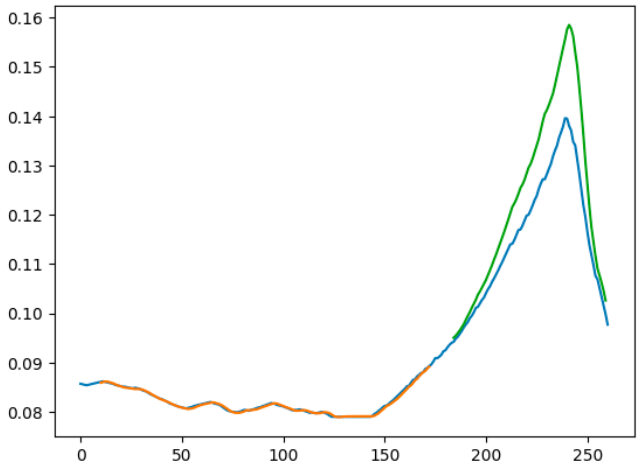


Data in blue, train data prediction in orange, test data prediction in green

TABLE X. LSTM – LINUX/UNIX

	Train	Test
RMSE	0.0003	0.0108
MAPE	0.9488%	14.2166%

And predictions for the Red Hat Enterprise Linux dataset:

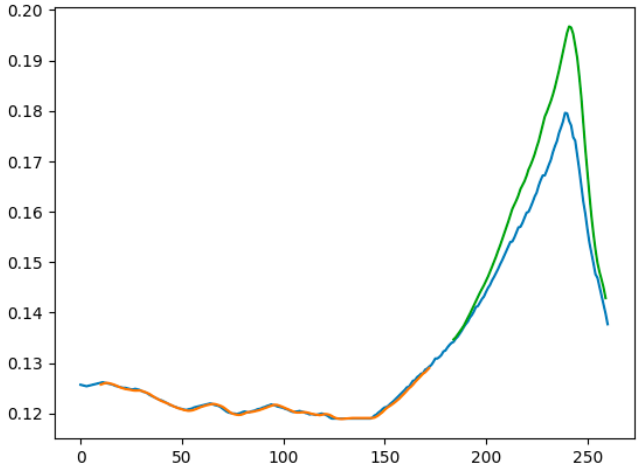


Data in blue, train data prediction in orange, test data prediction in green

TABLE XI. LSTM – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0002	0.0098
MAPE	0.2162%	6.4113%

Also predictions for the SUSE Linux dataset:



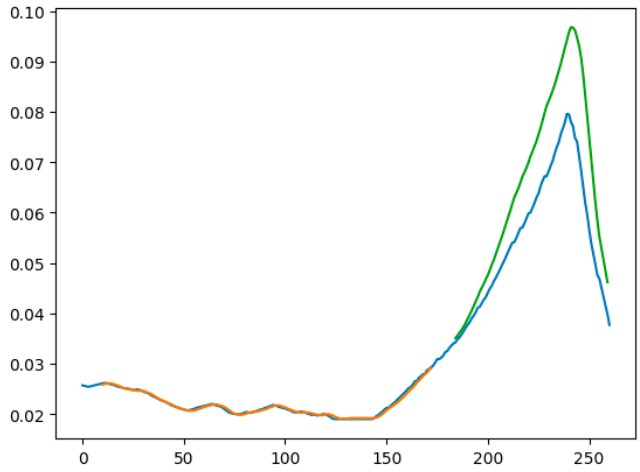
Data in blue, train data prediction in orange, test data prediction in green

TABLE XII. LSTM – SUSE LINUX

	Train	Test
RMSE	0.0003	0.0092
MAPE	0.1806%	4.4591%

Secondly, we undo the previous modifications and modify the number of epochs from 200 to 100. We can see the results also get worse.

We then make predictions for the Linux/UNIX dataset:

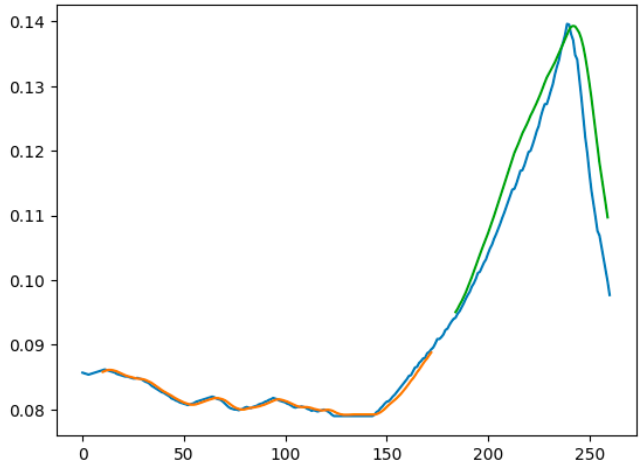


Data in blue, train data prediction in orange, test data prediction in green

TABLE XIII. LSTM – LINUX/UNIX

	Train	Test
RMSE	0.0003	0.0115
MAPE	1.0289%	16.0956%

And predictions for the Red Hat Enterprise Linux dataset:

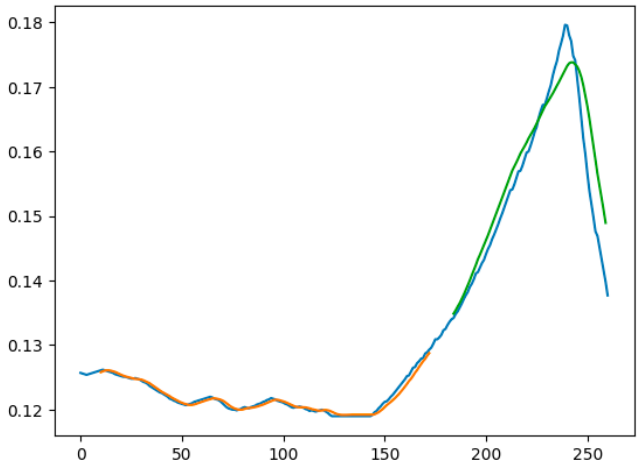


Data in blue, train data prediction in orange, test data prediction in green

TABLE XIV. LSTM – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0004	0.0061
MAPE	0.4299%	4.2078%

Also, predictions for the SUSE Linux dataset:



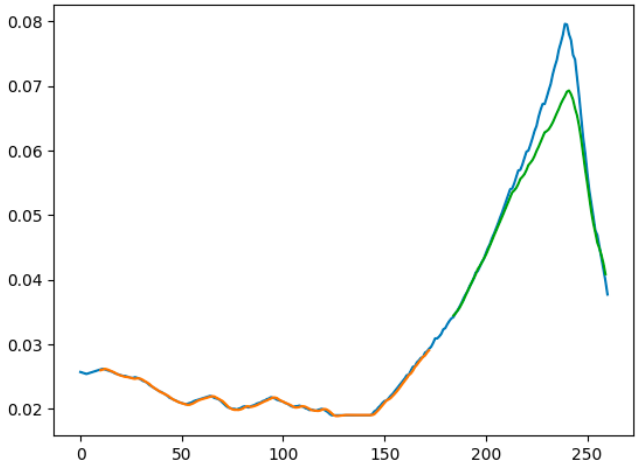
Data in blue, train data prediction in orange, test data prediction in green

TABLE XV. LSTM – SUSE LINUX

	Train	Test
RMSE	0.0004	0.0047
MAPE	0.2838%	2.2156%

Third, we also undo previous modifications and modify the batch size from 16 to 8. We can see the results are worse than the main scenario, but better than the two previous modifications we've made.

We then make predictions for the Linux/UNIX dataset:

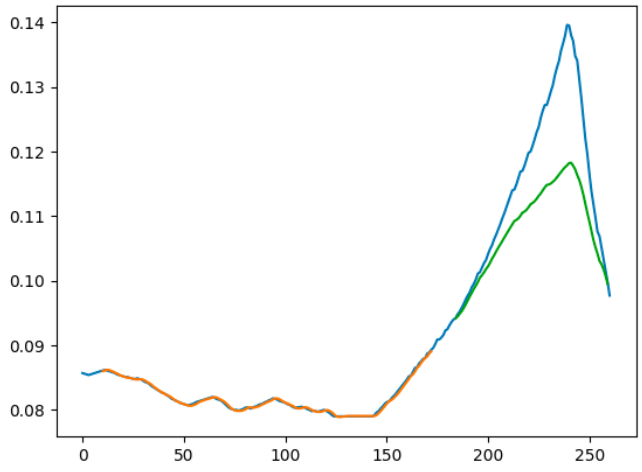


Data in blue, train data prediction in orange, test data prediction in green

TABLE XVI. LSTM – LINUX/UNIX

	Train	Test
RMSE	0.0002	0.0042
MAPE	0.6700%	4.1107%

And predictions for the Red Hat Enterprise Linux dataset:

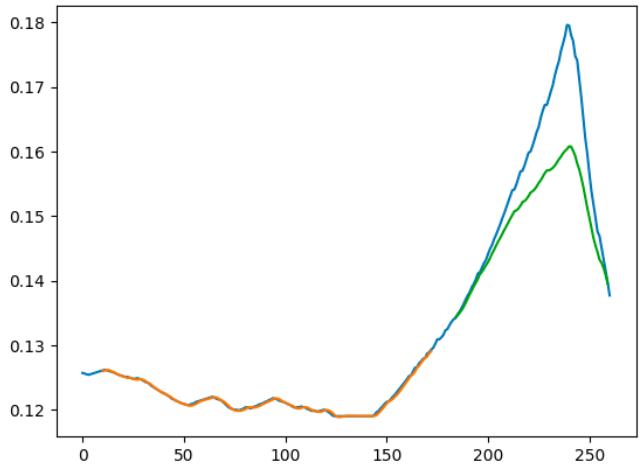


Data in blue, train data prediction in orange, test data prediction in green

TABLE XVII. LSTM – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0002	0.0099
MAPE	0.1822%	5.9932%

Also, predictions for the SUSE Linux dataset:



Data in blue, train data prediction in orange, test data prediction in green

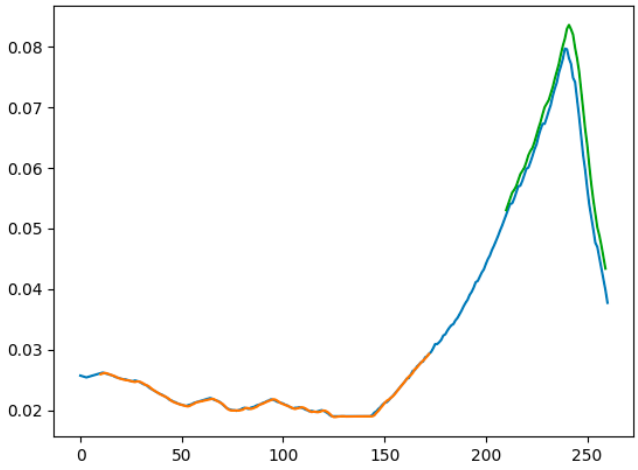
TABLE XVIII. LSTM – SUSE LINUX

	Train	Test
RMSE	0.0002	0.0085
MAPE	0.1182%	3.7936%

V. UPDATES AND FIXES

One problem encountered when checking the networks code was that we were using the test data as validation data in the CNNs implementations. This is obviously not ideal. So, we split the data differently specifically for the CNN. We continue with 67% of the data as the training set but we allocate 10% of the data as validation set and 23% of the data as the test set. The results are similar to the faulting version indicating that our error had little impact on the accuracy of the network:

First, we make predictions for the Linux/UNIX dataset:

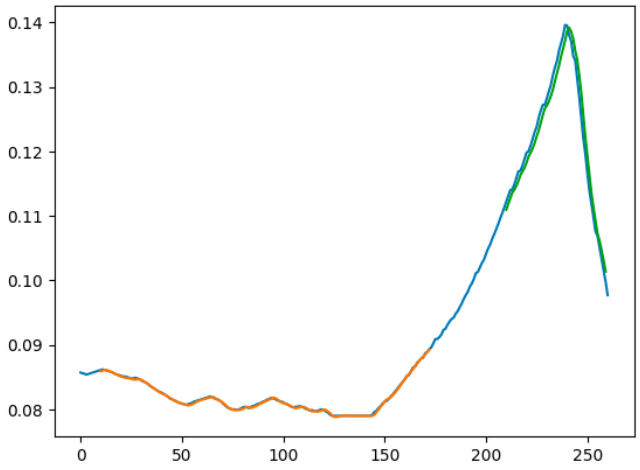


Data in blue, train data prediction in orange, test data prediction in green

TABLE XIX. CNN – LINUX/UNIX

	Train	Test
RMSE	0.0001	0.0038
MAPE	0.4257%	5.2087%

Secondly, we make predictions for the Red Hat Enterprise Linux dataset:



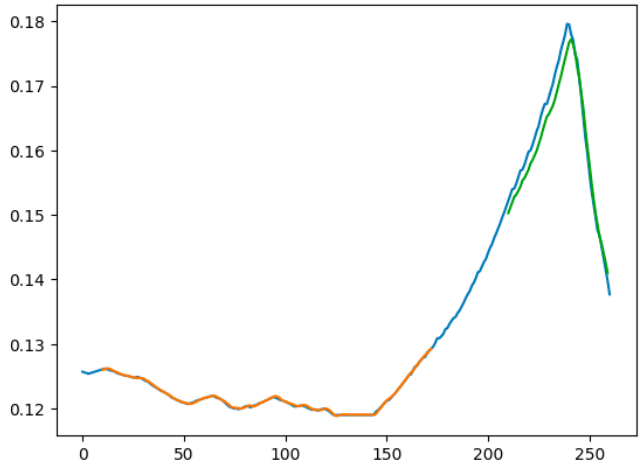
Data in blue, train data prediction in orange, test data prediction in green

TABLE XX. CNN – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0001	0.0019

	Train	Test
MAPE	0.1316%	1.3859%

Third, we make predictions for the SUSE Linux dataset:



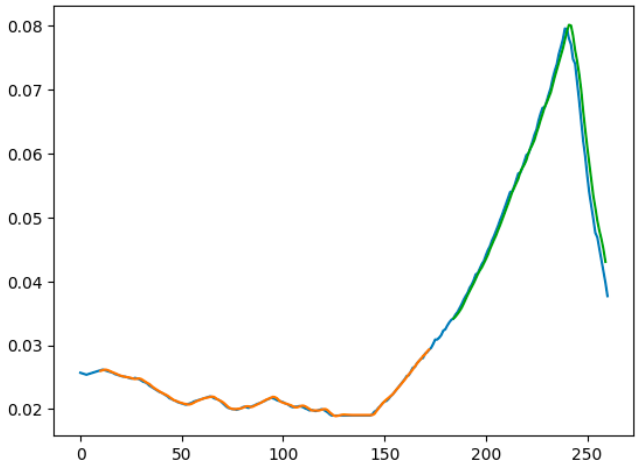
Data in blue, train data prediction in orange, test data prediction in green

TABLE XXI. CNN – SUSE LINUX

	Train	Test
RMSE	0.0001	0.0023
MAPE	0.0783%	1.2463%

Another problem found was that we were using the *validation_split* parameter of Keras library on the MLP model while also doing the test and training data separation manually. This can lead to overfitting because some of the test data could end being used as validation data. Here, our previous error had great impact on the accuracy of the model. As we removed the *validation_split* parameter the efficiency of the network improved dramatically:

First, we make predictions for the Linux/UNIX dataset:

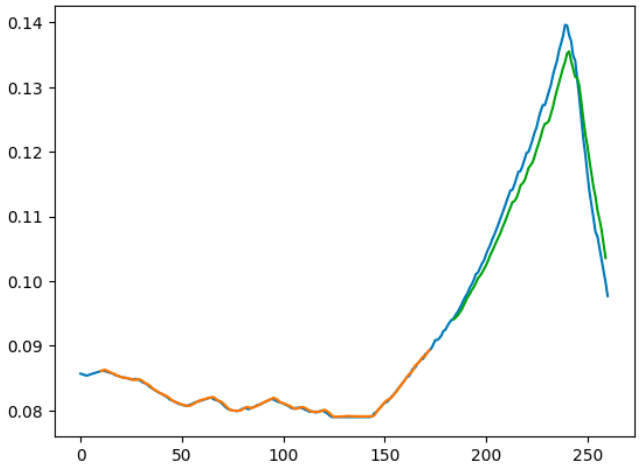


Data in blue, train data prediction in orange, test data prediction in green

TABLE XXII. MLP – LINUX/UNIX

	Train	Test
RMSE	0.0001	0.0019
MAPE	0.4963%	2.4668%

Secondly, we make predictions for the Red Hat Enterprise Linux dataset:

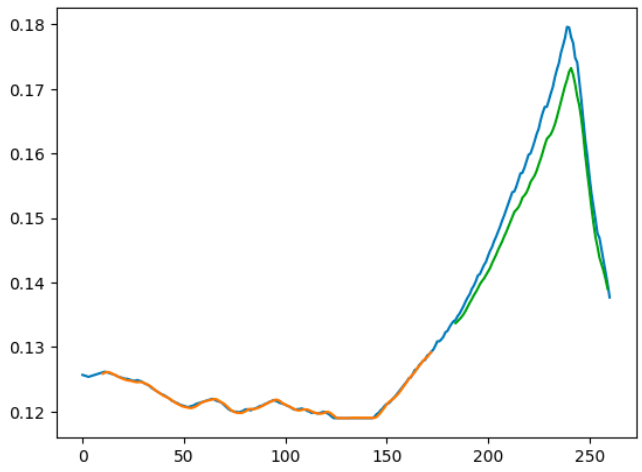


Data in blue, train data prediction in orange, test data prediction in green

TABLE XXIII. MLP – RED HAT ENTERPRISE LINUX

	Train	Test
RMSE	0.0001	0.0031
MAPE	0.1301%	2.3591%

Third, we make predictions for the SUSE Linux dataset:



Data in blue, train data prediction in orange, test data prediction in green

TABLE XXIV. MLP – SUSE LINUX

	Train	**Test**
RMSE	0.0002	0.0042
MAPE	0.1024%	2.2669%

VI. CONCLUSION

In this project, we trained and evaluated three deep learning models for predicting the spot prices of Linux-based EC2 instances. The three models LSTM, MLP and CNN performed well after our corrections, suggesting they are good choices for this task. Nonetheless, is imperative to train these models in larger sets of data so we can assess if they perform similarly. Our results can be used to help AWS users make informed decisions about the usage of Linux EC2 instances, based on their predicted spot prices.

Future work could involve usage of evolutionary algorithms to fine tune the hyperparameters of models, collecting more than 90 days' worth of data, exploring other deep learning architectures, using data from different regions, or incorporating additional features to improve the models' performance.

REFERENCES

[1] *Agarwal, Sonam, Ashish Kumar Mishra, and Dharmendra K. Yadav. "Forecasting price of amazon spot instances using neural networks." Int. J. Appl. Eng. Res 12.20 (2017): 10276-10283.*