Asset price prediction

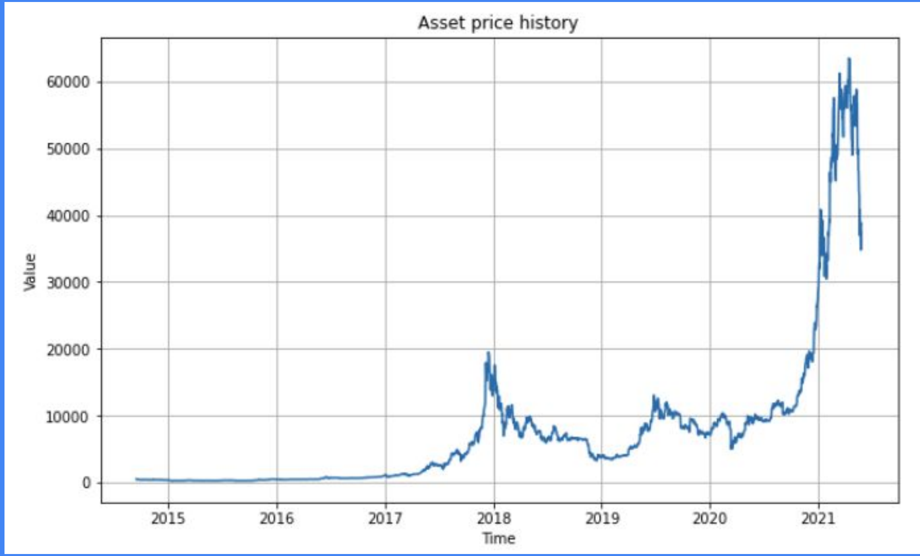
using Python, NumPy, Deep Learning and Tensorflow

The Plan

- Obtain and prepare the Dataset
- Basic forecasting implementation
    - Naive
    - Moving Average
- Deep Learning implementation
    - Dataset preparation
    - Intro to Deep Learning and Tensorflow
    - Creating the model
        - Training hyperparameters
    - Training the model & forecasting
- Summary & feedback

Disclaimer

This is not a financial advice nor a promotion of trading or any asset. Just sharing my interests, knowledge and experience



Dataset: Timeseries

Exercise: download and explore dataset

- Check you are a member of Workshops chat
- Fork & clone Github repo, or use:
- Exercises colab notebook
- Answers colab notebook

- Pick the asset to work with:
    - Heineken, Bitcoin, Dogecoin, Gold, Starbucks, Tesla...
- Go to finance.yahoo.com and download its trading history csv file
- Place it under “**sample_data**” directory
- Explore the dataset
    - ‘Date’ & ‘Close’ columns

Exercise 1: read and prepare data

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600

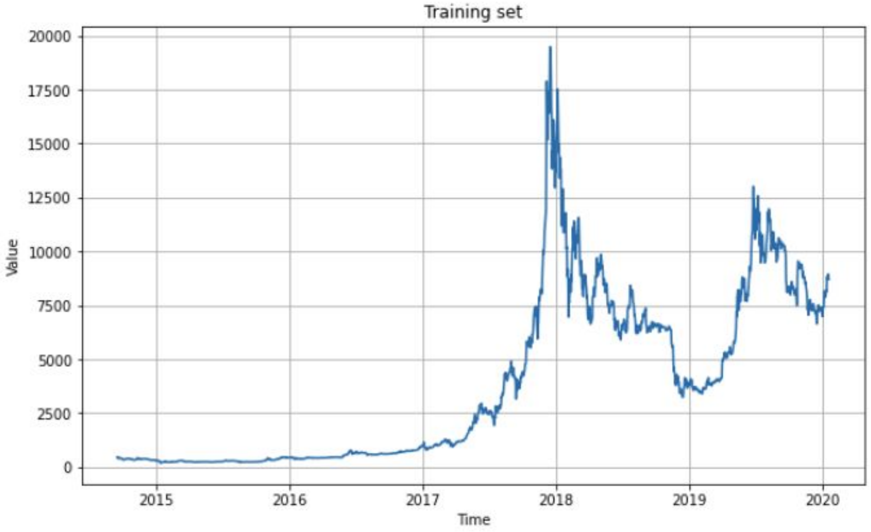
Read data from CSV:

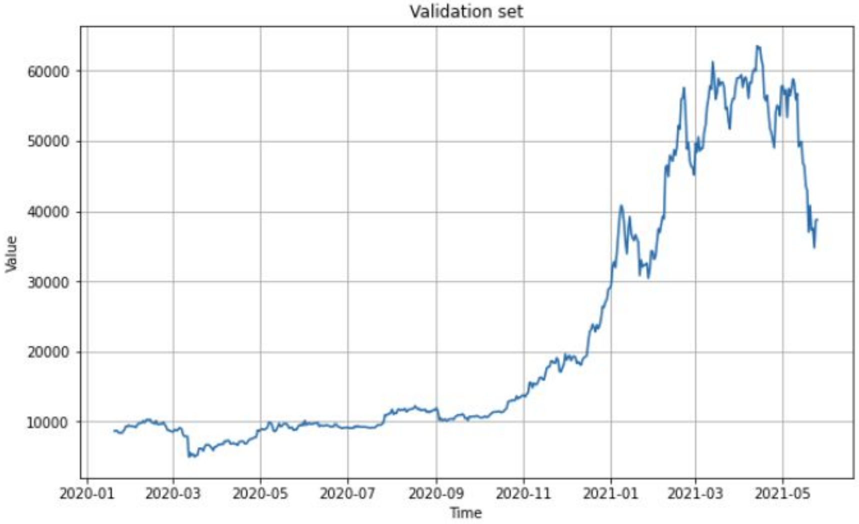
```
with open(filepath) as csvfile:
    reader = csv.reader(csvfile,delimiter=',')
    next(reader) # skip the header
    for row in reader: # iterate over rows
        x.append(row[0])
        y.append(row[1])
return x, y
```

Split into training and validation set:

```
split_time = time_len * 0.8
time_train = time[:split_time]
x_train = ...
time_valid = time[split_time:]
x_valid = ...
```

Dataset: plot results





Basic forecasting

Exercise 2: Naive implementation & metrics

- Used as a baseline to compare against other models
- Often is hard to beat
- Basic idea: today's forecast = yesterday's data

```
forecast = series[start - 1:end]
```

- Metrics to evaluate performance:
    - MSE - Mean Squared Error

    ```
    keras.metrics.mean_squared_error(validation, forecast)
    ```

    - MAE - Mean Absolute Error

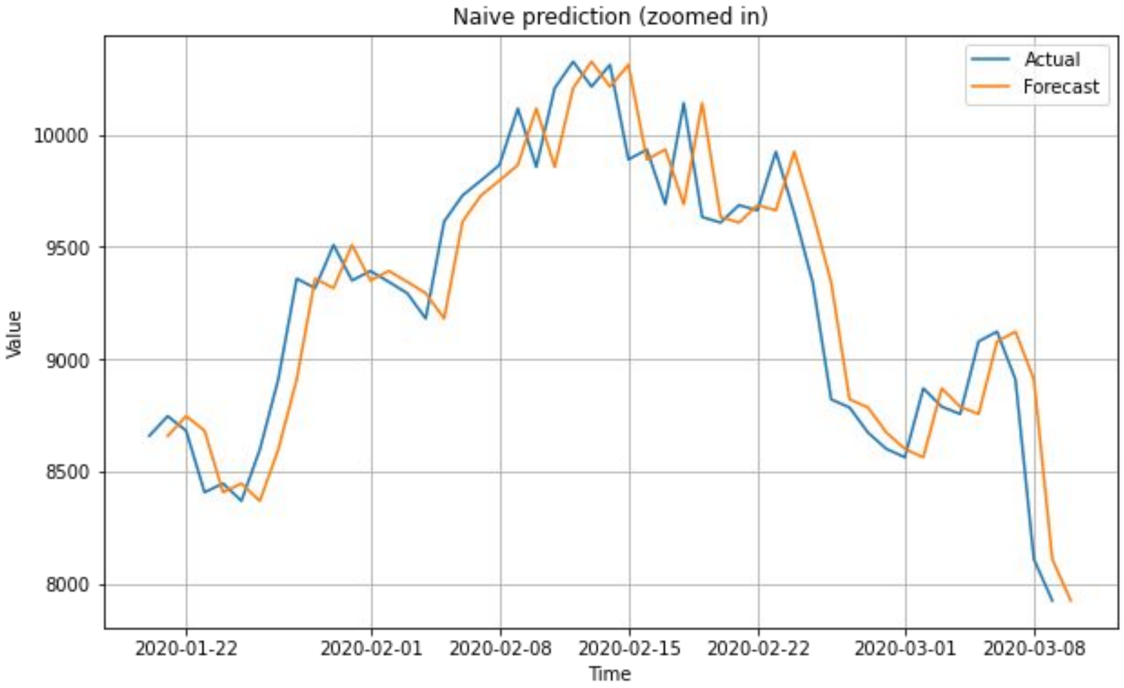
    ```
    keras.metrics.mean_absolute_error(validation, forecast)
    ```

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Naive implementation: results

MSE:
1567890.9365762253
MAE:
668.2866531270491



Exercise 3: Moving Average

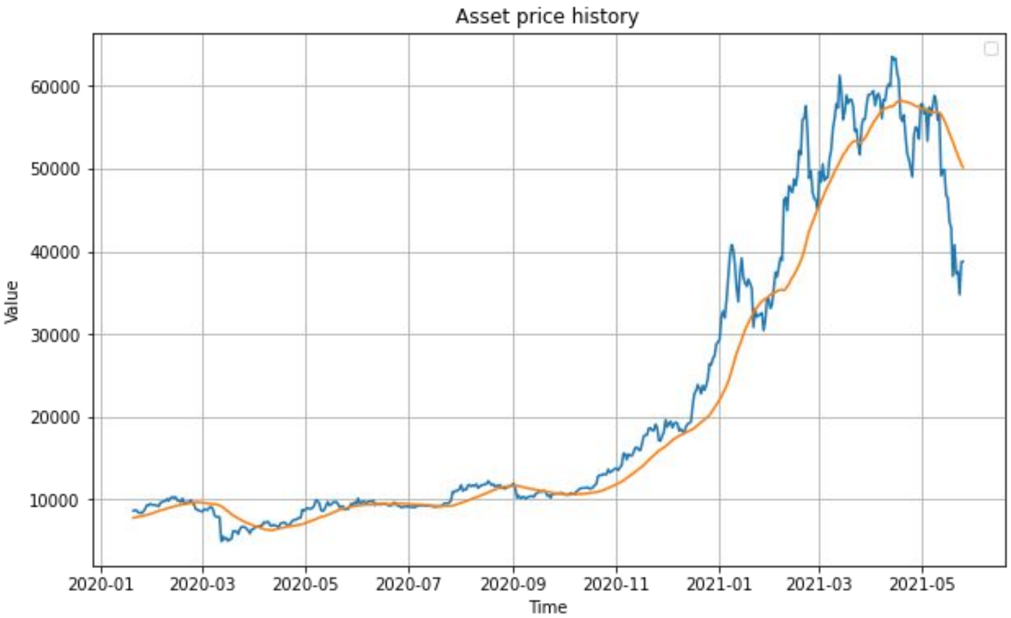
- Simple forecasting method
- Calculates **average** of values over a **fixed period** of time (**averaging window**)
- Usually performs worse than Naive Forecast
- Using window of 30 days:

```
for time in range(series_range - 30):
   mean = series[time:time + 30]mean()
   forecast.append(mean)
return forecast
```

Moving Average: results

Moving Average:
MSE:
17180985.74535265
MAE:
2472.22143907015

Naive:
MSE:
1567890.9365762253
MAE:
668.2866531270491



Deep Learning time!

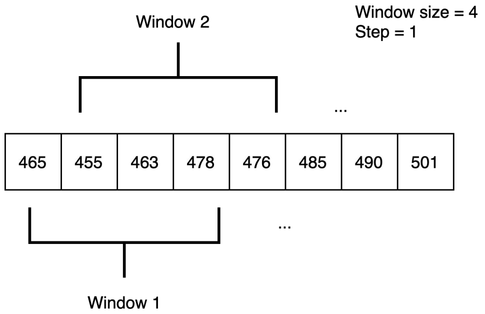
Exercise 1: Prepare dataset for training

- As before:
    - Use Github repo, or:
    - Use colab exercises and answer notebooks
    - Specify and load CSV file
    - Split into training and validation sets

- **Normalize** training and validation sets

```
max_val = np.max(np.abs(my_arr),axis=0)

my_arr /= max_val
```

- *Split dataset into **smaller windows** of 20 items (input size)

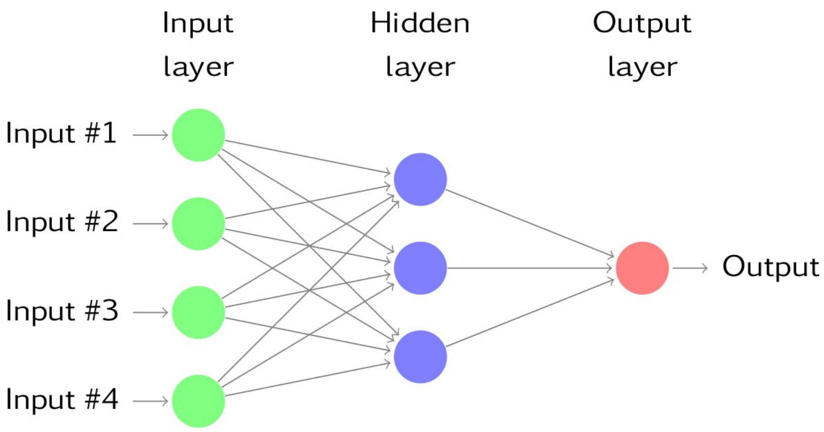


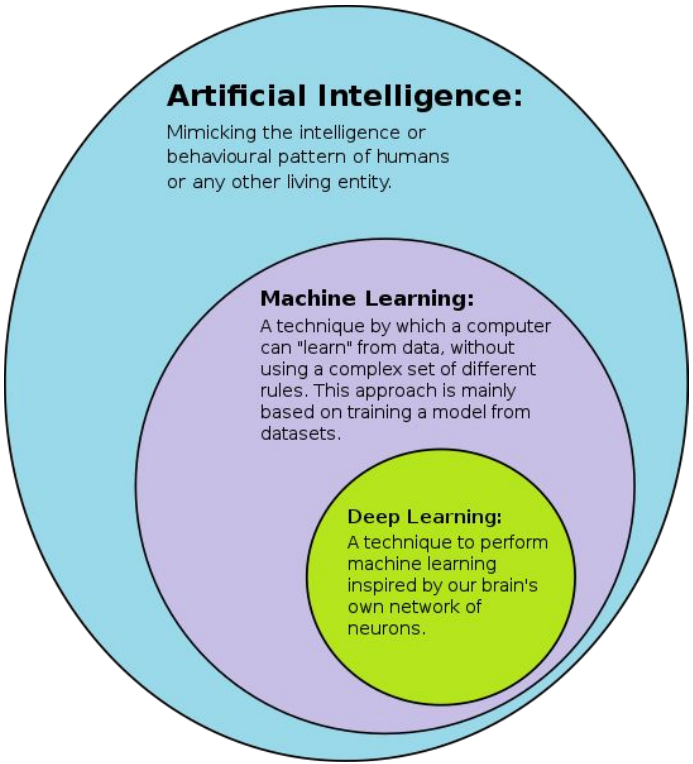
- ***Shuffle** the windows
- *Group in **batches** of 32 windows for parallel training

*See `def windowed_dataset` for more details

Intro to Deep Learning and TensorFlow

- Subset of ML based on Artificial Neural Networks





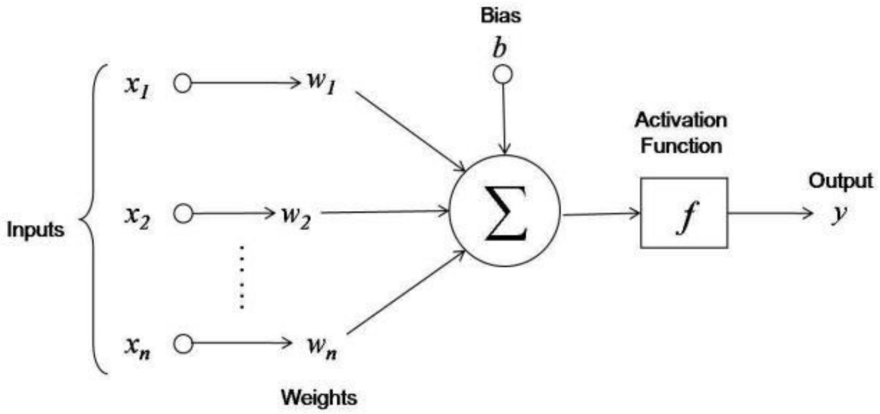
- Applied in many fields:
  - Computer vision
  - Natural language processing
  - Sequence models

Intro to Deep Learning and TensorFlow

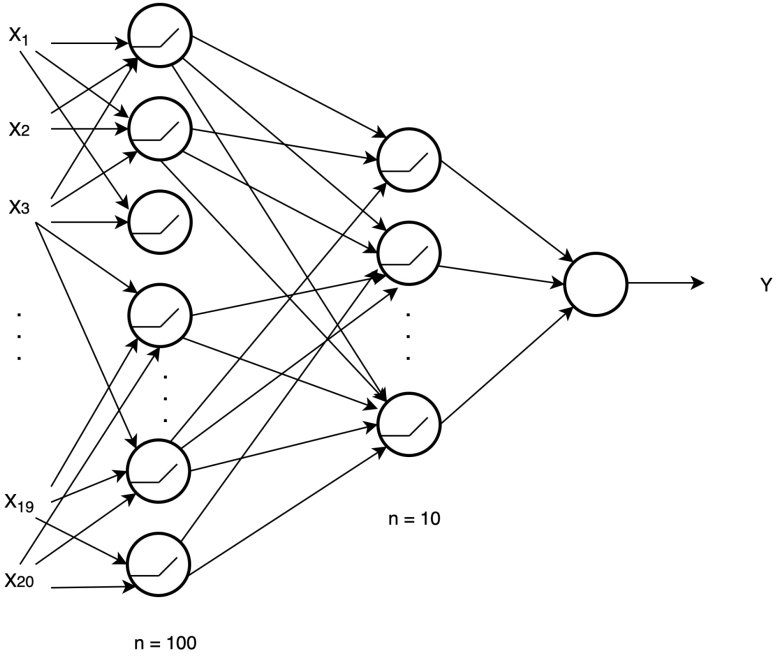
- Open source framework to easily create, train & use DL models
- Easy prototyping, deploy & use in Production
- Ecosystem:
  - TensorFlow.js
  - TensorFlow Lite
  - TensorFlow Extended (TFX)
  - TensorFlow Hub
  - TensorFlow Datasets
  - Colab
  - And many more

Deep Learning model: Perceptron

$$Y = f(WX + b)$$

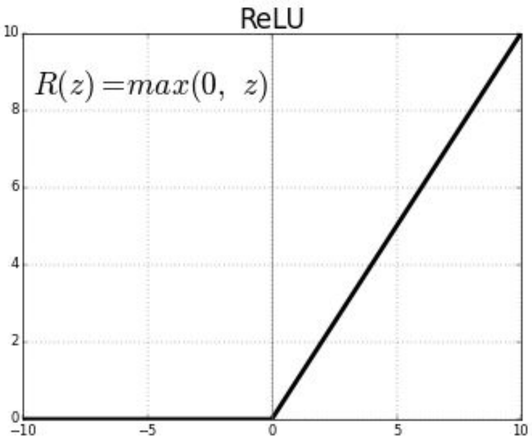


Deep Learning model: Architecture



Rectified Linear Unit

- Easy to implement and optimize
- Outputs [0; ∞]



Exercise 2: Create a DL model

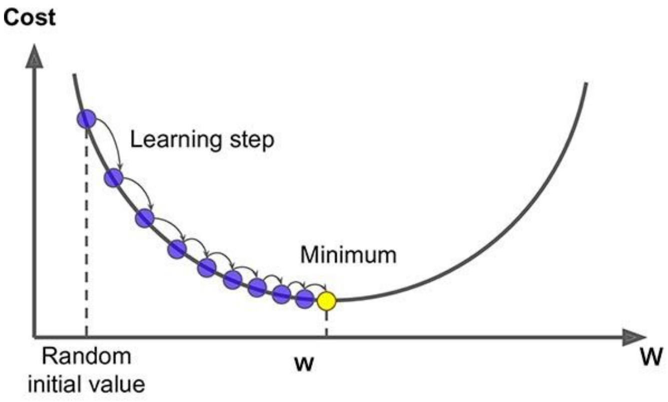
- Do not forget to specify your dataset file
- We will use Tensorflow Sequential API to construct our model
- Example: a simple two-layer network with 10-1 neurons:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(10, input_shape=[input_size], activation="relu"),
    tf.keras.layers.Dense(1)
])
model.summary()
```

Create a DL model: results

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 100)               2100
_________________________________________________________________
dense_37 (Dense)             (None, 10)                1010
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 11
=================================================================
Total params: 3,121
Trainable params: 3,121
Non-trainable params: 0
```

Training the model



Compile model params:

- Loss function
- Optimizer function
  - Learning rate
  - Momentum

```
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-6, momentum=0.9))
```

Fit the model on a dataset:

- Dataset to train on
- # of Epochs to train

```
model.fit(dataset,epochs=500,verbose=2)
```

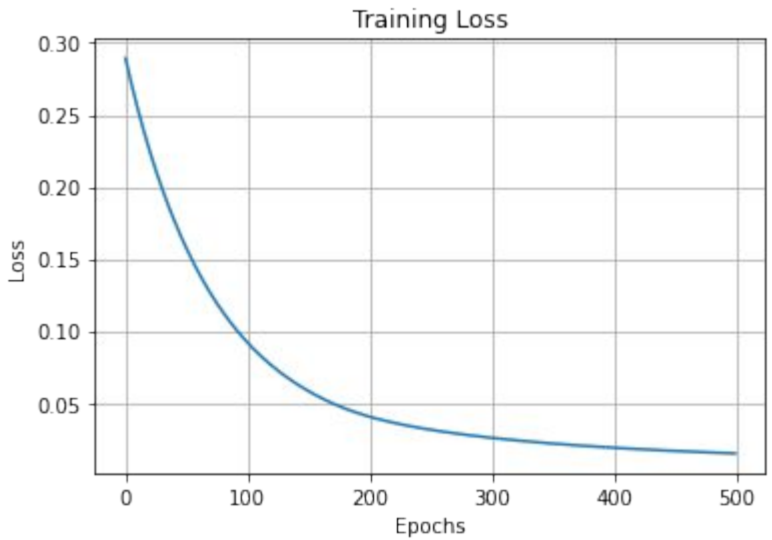
Exercise 3: train the model

Example: compile & train for 50 epochs:

```
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-5, momentum=0.95))

model.fit(dataset,epochs=50,verbose=2)
```

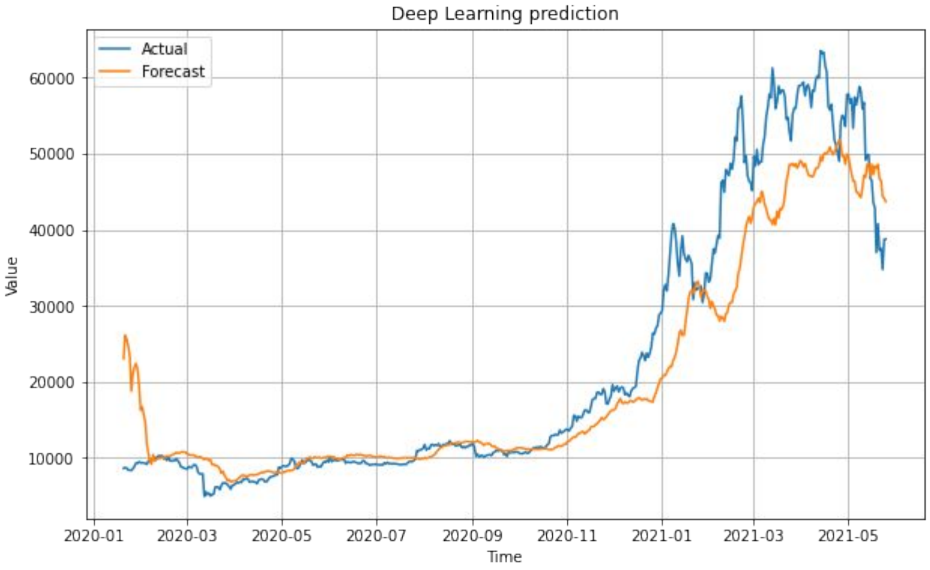
Training the model: results

```
Epoch 1/500

61/61 - 0s - loss: 0.0372

Epoch 2/500

61/61 - 0s - loss: 0.0364

...

Epoch 499/500

61/61 - 0s - loss: 0.0020

Epoch 500/500

 61/61 - 0s - loss: 0.0020
```



Forecasting: results

MSE:
43001720.0
MAE:
4110.766



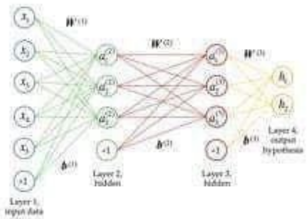
Summary

- Which model performed better for your data?
- Possible improvements:
  - Play around with **hyperparameters**:
    - # of epochs
    - Adjust learning rate
    - # of neurons/layers
  - Use more **advanced** techniques:
    - Dropout regularization
    - LSTM (Long short-term memory) layers
    - Convolution layers

Feedback

Please let me know what you think about the topic

Thanks!

	A parrot 	Machine learning algorithm 
Learns random phrases	✅	✅
Doesn't understand shit about what it learns	✅	✅
Occasionally speaks nonsense	✅	✅
Is a cute birdie parrot	✅	❌