Université du Luxembourg

Bachelor en Sciences et Ingénierie

Mathématiques expérimentales 4

2020-2021

Stocktrading with machine learning models

Authors: Gil Moes & Mirza Muharemovic

Tutor: M. George Kerchev

Contents

1 Introduction

The stock market is one of the main tools for industries to expand and thus plays an important role in the world of economy. In fact companies listed in the stock market can issue stocks of their company to raise funds from the public and invest these funds in their growth. If the company manages to raise it's worth, the stocks also increase in value which is the main goal of the investors. Since this is not always the case, investors have to be careful when choosing to buy stocks and consider the future development of the stock's value.

To make these decisions, researchers have spent years on constructing economical models which can be used to analyse specific data sets to predict their evolution. Although these models are not only meant for economical applications, we're going to focus solely on stocks for this project.

We're going to talk about machine learning models and use these to make predictions regarding the value of certain stocks. These models can be seen as an application of artificial intelligence which provides systems that improve by analysing data sets and not by getting further input from the user.
These models access parts of a data set and use the provided information to find patterns in the data to make predictions.
Then by analysing the next batch data, the program tries to improve the predictions it has made using the first batch. This process is repeated with the goal being that the program can make precise predictions.

To analyse these models we chose to take the stocks of AMD (Advanced Micro Devices) as an example. The AMD stock values were particularly interesting as AMD has managed to catch up to its competitors which translates to a drastic increase in its stock values over the last 5 years.

2 Regression line

2.1 Relation between variables

The covariance and the linear correlation coefficient are measurement instruments that are used to determine the existence of a certain linear relation between variables x and y.
Note that these instruments only describe a relation between variables where we can suppose that they are somehow linked. They don't prove the existence of a relation.

2.2 Covariance

The covariance measures the joint variation of the modalities of two variables. It describes the common variation of the modalities of two variables around their respective average.

$$\rho_{xy} = \tfrac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y}) = (\frac{1}{N}\sum_{i=1}^{N} x_i \cdot y_i) - \overline{xy}$$

$$\bar{x} = \tfrac{1}{N}\sum_{i=1}^{N} x_i \text{ same for } \bar{y}$$

(N observations ; $\bar{x}$ and $\bar{y}$ are the arithmetic average of the x_i's and y_i's ; x_i are the real values ; y_i are the observed values)

A positive covariance indicates that the variation of the modalities of the two variables is done in the same direction.
In this case we talk about a 'positive linear dependence'.
On the other hand, if they evolve in an opposite way, we talk about a 'negative linear dependence'.
A covariance close to 0 allows us to conclude that there is no linear independence.

2.3 The linear correlation coefficient

The linear correlation coefficient allows us to draw the same conclusion as with the covariance considering the linear dependence of the modalities of two variables. The advantage of the linear correlation coefficient is that it is normalized and takes values between -1 and 1.

$$\rho_{xy} = \tfrac{\sigma_{xy}}{\sigma_x \cdot \sigma_y}$$

$$\sigma_x = \sqrt{(\tfrac{1}{N}\sum_{i=1}^{N} x_i^2) - \bar{x}^2} \quad \text{same for } \sigma_y$$

Note: σ represents the standard deviation.

The order of magnitude of ρ_{xy} gives us indications on the intensity of the linear dependence between the modalities of two variables.

Linear correlation coefficient	Intensity of the linear correlation
$\lvert\rho_{xy}\rvert \leq 0.2$	very weak correlation (nearly inexistent)
$0.2 < \lvert\rho_{xy}\rvert \leq 0.5$	weak correlation
$0.5 < \lvert\rho_{xy}\rvert \leq 0.7$	average correlation
$0.7 < \lvert\rho_{xy}\rvert \leq 0.9$	higher correlation
$0.9 < \lvert\rho_{xy}\rvert \leq 1$	very high correlation

Remark:

Later on we will see that the relation described by the equation of the regression line isn't exact but only approximated.
To see how precise the estimated equation is we can calculate the determination coefficient $R^2 = \rho_{xy}^2 \in]0, 1[$).
A low value of R^2 suggests that the estimated relation doesn't adjust very well to the observed data. A high value of R^2 suggests that the estimated relation is adequate.

2.4 Cloud of points

It is useful to graphically represent the point couples (x_i, y_i). Using the graph one can observe if there exists a linear relation.
The equation that describes such a relation between two variables is of the form $y = \alpha \cdot x + \beta$.
Note that the points of the cloud can be found above the straight line and under it and not necessarily on the straight line.
Therefore there exists a difference for all estimated values $\hat{y}_i$ $(= \alpha \cdot x_i + \beta)$ and for all observed values y_i such that $e_i = y_i - \hat{y}_i$.
In order to decrease the distance between these estimated values $\hat{y}_i$ and the real observed values y_i, the parameters alpha and beta need to be determined in a way such that the sum of the square of the differences is minimal ($S = \sum\limits_{i=1}^{N} e_i^2$).

2.5 Mathematical development

We rewrite S:

$$S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} (y_i - \alpha \cdot x_i - \beta)^2$$

In order to find the values of α and β that minimize the expression S, we first calculate the partial derivatives of α and β respectively.

$$\begin{cases} S'_\alpha = 0 \\ S'_\beta = 0 \end{cases} \iff \begin{cases} S'_\alpha = -2\sum\limits_{i=1}^{N}(x_i y_i - \alpha x_i^2 - \beta x_i) = 0 \\ S'_\beta = -2\sum\limits_{i=1}^{N}(y_i - \alpha x_i - \beta) = 0 \end{cases}$$

This gives us the following system of equations:

$$\begin{cases} \sum_{i=1}^{N}(x_iy_i - \alpha x_i^2 - \beta x_i) = 0 \\ \sum_{i=1}^{N}(y_i - \alpha x_i - \beta) = 0 \end{cases} \iff \begin{cases} \sum_{i=1}^{N} x_iy_i - \alpha \sum_{i=1}^{N} x_i^2 - \beta \sum_{i=1}^{N} x_i = 0 \ \ (1) \\ \sum_{i=1}^{N} y_i - \alpha \sum_{i=1}^{N} x_i - N\beta = 0 \ \ (2) \end{cases}$$

Now we transform (2) to express the parameter β with α:

$$\beta = \frac{\sum_{i=1}^{N} y_i - \alpha \sum_{i=1}^{N} x_i}{N} = \bar{y} - \alpha\bar{x}$$

We observe that the regression line passes through the point with coordinates $(\bar{x}, \bar{y})$.
We now replace β in (1):

$$\sum_{i=1}^{N} x_iy_i - \alpha \sum_{i=1}^{N} x_i^2 - (\bar{y} - \alpha\bar{x}) \sum_{i=1}^{N} x_i = 0$$

By isolating α we get the following expression:

$$\alpha = \frac{\sum_{i=1}^{N} x_iy_i - \bar{y} \sum_{i=1}^{N} x_i}{\sum_{i=1}^{N} x_i^2 - \bar{x} \sum_{i=1}^{N} x_i}$$

In order to be able to simplify this expression later on we will multiply the nominator and denominator with $\frac{1}{N}$, hence we get:

$$\alpha = \frac{\frac{1}{N}(\sum_{i=1}^{N} x_iy_i - \bar{y} \sum_{i=1}^{N} x_i)}{\frac{1}{N}(\sum_{i=1}^{N} x_i^2 - \bar{x} \sum_{i=1}^{N} x_i)} = \frac{\frac{1}{N} \sum_{i=1}^{N} x_iy_i - \bar{y}\frac{1}{N} \sum_{i=1}^{N} x_i}{\frac{1}{N} \sum_{i=1}^{N} x_i^2 - \bar{x}\frac{1}{N} \sum_{i=1}^{N} x_i}$$

2.6 Regression line equation

The equation of the regression line is given by $y = \alpha \cdot x + \beta$.

Note that

$$\alpha = \frac{(\frac{1}{N} \sum_{i=1}^{N} x_i \cdot y_i) - \overline{xy}}{(\frac{1}{N} \sum_{i=1}^{N} x_i^2) - \bar{x}^2} \text{ and } \beta = \bar{y} - \alpha \cdot \bar{x} \quad .$$

2.7 Program Output

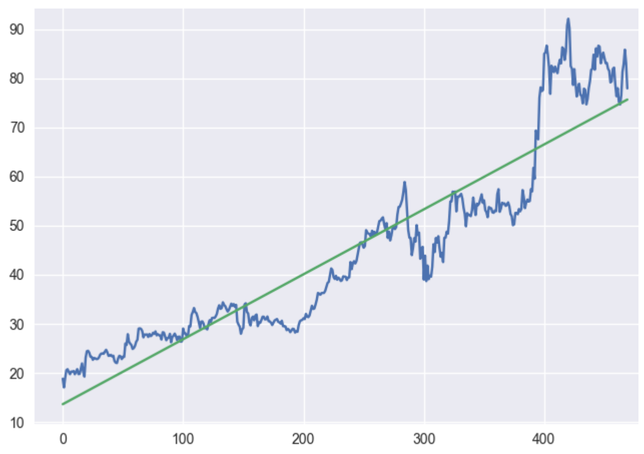


Figure 1: Regression line

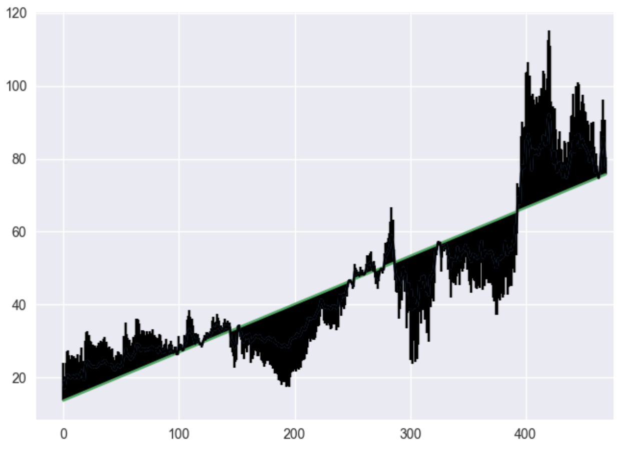


Figure 2: Regression line with error bars

2.7.1 Observations

We observed that by considering a too large time period the R^2 coefficient indicates that the estimated relation doesn't adjust well to the data.
In *Figure 1* we used the time period of the stock values of AMD from 01.01.2019 to 10.11.2020 and we got $R^2 = 0.8633751494889846$. Hence 86% of the variation of stocks are due to time and 14% are related to other factors.
In *Figure 2* we also included the error bars e_i. The error bars are vertical lines going from the regression line to the y_is and the length is mirrored in the opposite direction, i.e. they represent twice the difference between the real values and the estimated value.

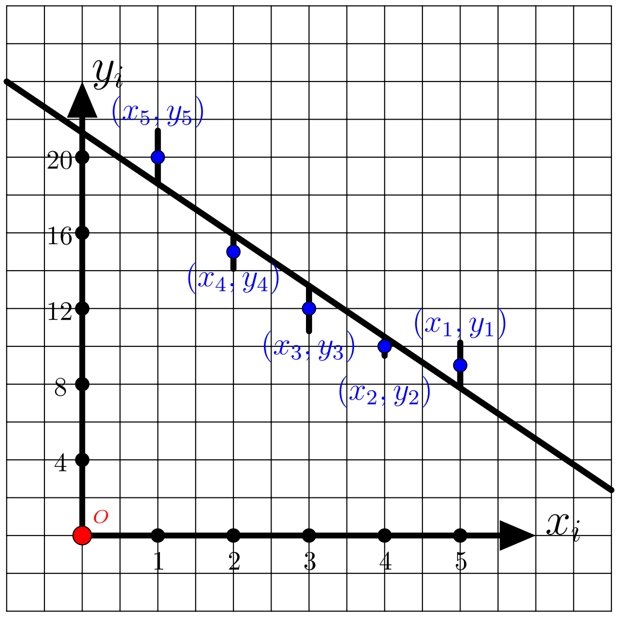
2.7.2 Other example

There is a relation between the price of a product and the demand of that product:

We consider the following statistical informations:

Product price in € (x_i)	Annual demand (y_i)
5	9
4	10
3	12
2	15
1	20

Graphical representation of the cloud of points with the regression line and error bars:



Calculation of the regression line:

We need to determine the following:

$$y = \alpha \cdot x + \beta$$

$$\alpha = \frac{(\frac{1}{N}\sum_{i=1}^{N} x_i \cdot y_i) - \overline{xy}}{(\frac{1}{N}\sum_{i=1}^{N} x_i^2) - \bar{x}^2} \text{ and } \beta = \bar{y} - \alpha \cdot \bar{x}$$

$$\bar{x} = \tfrac{1}{N}\sum x_i \text{ and } \bar{y} = \tfrac{1}{N}\sum y_i$$

Here $N = 5$

$\sum_{i=1}^{N} x_i \cdot y_i = 5 \cdot 9 + 4 \cdot 10 + 3 \cdot 12 + 2 \cdot 15 + 1 \cdot 20 = 171$

$\sum_{i=1}^{N} x_i^2 = 5^2 + 4^2 + 3^2 + 2^2 + 1^2 = 55$

$\bar{x} = \tfrac{1}{N}\sum x_i = \dfrac{1}{5} \cdot (1 + 2 + 3 + 4 + 5) = 3$

$\bar{y} = \tfrac{1}{N}\sum y_i = \dfrac{1}{5} \cdot (9 + 10 + 12 + 15 + 20) = \dfrac{66}{5}$

Hence:

$\alpha = \frac{\frac{1}{5}\cdot 171 - 3 \cdot \frac{66}{5}}{\frac{1}{5}\cdot 55 - 3^2} = \frac{-27}{10} = -2.7$

$\beta = \frac{66}{5} - (-2.7) \cdot 3 = \frac{213}{10} = 21.3$

Hence the regression line is given by: $y = -2.7x + 21.3$

Prediction using the regression line:

If the price of a product were 6 € $(x_6 = 6)$, what would be the demand?
Answer: $y_6 = -2.7 \cdot 6 + 21.3 = 5.1 \approx 5$

By considering the errors:
$e_1 = y_1 - \hat{y}_1 = 20 - (\alpha \cdot x_1 + \beta) = 20 - (-2.7 \cdot 1 + 21.3) = 1.4$
$e_2 = y_2 - \hat{y}_2 = 15 - (\alpha \cdot x_2 + \beta) = 15 - (-2.7 \cdot 2 + 21.3) = -0.9$
$e_3 = y_3 - \hat{y}_3 = 12 - (\alpha \cdot x_3 + \beta) = 12 - (-2.7 \cdot 3 + 21.3) = -1.2$
$e_4 = y_4 - \hat{y}_4 = 10 - (\alpha \cdot x_4 + \beta) = 10 - (-2.7 \cdot 4 + 21.3) = -0.5$
$e_5 = y_5 - \hat{y}_5 = 9 - (\alpha \cdot x_5 + \beta) = 9 - (-2.7 \cdot 5 + 21.3) = 1.2$

Hence for 6 € the estimated demand of the product at that price
is $5.1 + error \mid -1.2 \leq error \leq 1.4$ for the given statistical information.

R^2-coefficient:

$$R^2 = \rho_{xy}^2 = (\tfrac{\sigma_{xy}}{\sigma_x \sigma_y})^2 = \left(\frac{(\frac{1}{N}\sum_{i=1}^{N} x_i \cdot y_i) - \overline{xy}}{\sqrt{(\frac{1}{N}\sum_{i=1}^{N} x_i^2) - \bar{x}^2} \cdot \sqrt{(\frac{1}{N}\sum_{i=1}^{N} y_i^2) - \bar{y}^2}}\right)^2 = \left(\frac{-5.4}{\sqrt{2} \cdot \frac{\sqrt{394}}{5}}\right)^2 \approx (-0.96)^2 = 0.9216$$

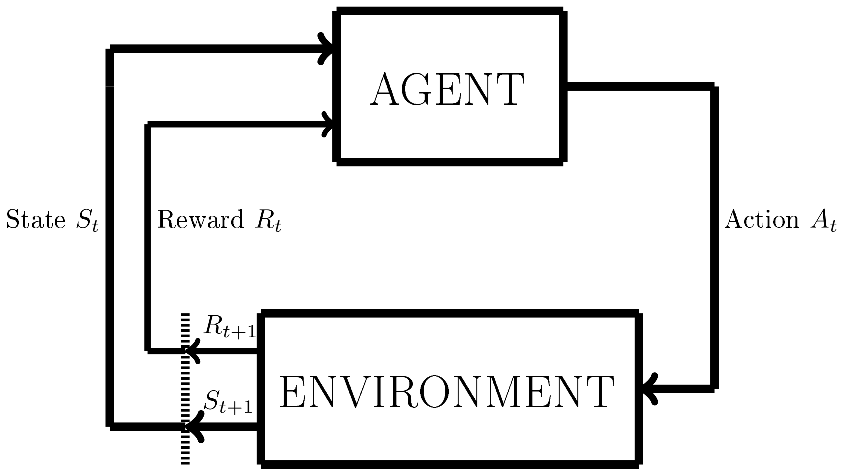
Hence: $\approx 92\%$ of the variation of the demand is due to the price.

(Note: $\sigma_{xy} = -5.4$, thus we have a negative linear dependence and $|\rho_{xy}| \approx |-0.96| = 0.96$ indicates that we have a very high correlation)

3 Brief introduction to finite Markov decision processes (MDP)

In mathematics, a Markov decision process (MDP) is a discrete-time randomly determined control process. 'Discrete time' means that the values of variables are viewed as occurring at distinct, separate 'points in time'. The MDP makes it possible to model decision making in situations where the outcomes are partly random and partly under the control of a decision maker (agent). MDPs are used to study optimization problems that are solvable with dynamic programming and reinforcement learning. They are applied in disciplines like robotics, automatic control, economics and manufacturing.

MDPs solve a problem by learning from interactions. We have an 'Agent' that is basically a learner and decision maker. The agent interacts continually with its environment. The agent selects actions and the environment responds to them by presenting new situations to the agent. Apart from providing new situations the environment also gives rise to rewards that the agent wants to maximize by taking corresponding actions.



The agent and environment interact with time steps t $(= 0, 1, 2, 3, ...)$. At each step, the agent receives a representation of the environment's state (S_t) and based on the state chooses an action A_t. This way we get a state-action pair (S_t, A_t). Based on the action A_t taken by the agent from the state S_t it then receives a numerical reward $R_{t+1} \in \mathcal{R}$ and therefore finds itself in a new state S_{t+1}.
One can think of this process as a function that maps state-action pairs to rewards.
Hence at each time t, we have: $f(S_t, A_t) = R_{t+1}$.
The MDP gives rise to a sequence like: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$
In the case of a finite MDP the sets of states, actions, and rewards ($\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$) all have a finite number of elements.

Then the random variables R_t and S_t have well defined discrete probability distributions where the only dependencies are the preceding state and action. Hence: All possible values that one could assign to S_t and R_t have a certain associated probability. Thus for some random variables $s' \in \mathcal{S}$ and $r \in \mathcal{R}$ there is a probability that $S_t = s'$ and $R_t = r$ if certain preceding values of state ($s \in \mathcal{S}$) and action ($a \in \mathcal{A}$) are given.
The probability of the transition to state s' and reward r from taking action a in state s is then:

$$p(s', r|s, a) = \mathbb{P}\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \ \forall s', s \in \mathcal{S},\ r \in \mathcal{R} \text{ and } a \in \mathcal{A}$$

(p defines the dynamics of the MDP and $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$ is a deterministic function) where $|$ tells us that p specifies a probability distribution for each choice of s and a:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \ \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

The probability described by p completely characterizes the environment's dynamics. Hence the probability of each possible value for S_t and R_t depends on the immediately preceding state and action S_{t-1} and A_{t-1}, but it depends not on all earlier states and actions. This is best shown by a restriction on the state. The state needs to include all the information about every aspect of the past agent-environment interactions that will make a difference in the future. If this condition is fulfilled the state is said to have the Markov property.

Lastly note that one can also compute the expected rewards for state–action pairs as a two-argument function $r : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ (1) or the expected rewards for state–action–next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow \mathbb{R}$ (2), given by the following calculations:

- (1) $r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$
- (2) $r(s, a, s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \cdot \frac{p(s', r|s, a)}{p(s'|s, a)}$

Remark: MDPs are a stepping stone towards RNN and LSTM so we won't go into more details. For more details or examples of application, take a look at the reference [18].

4 Recurrent neural network

4.1 Definitions

4.1.1 Machine learning

The term Machine learning is used to describe computer systems that learn from data, i.e. algorithms that are programmed to recognize patterns in data and use these to make predictions once they are fed new data.

4.1.2 Deep learning

Deep learning is often used interchangeably with machine learning but it is important to note that they do not mean the same thing. The term deep learning describes algorithms with a layered structure which are denoted as artificial neural network. The goal of deep learning is to get to the point where the algorithms analyse data and draw conclusions similar to how the human brain does.
Hence deep learning is in fact a sub-category of machine learning.

4.1.3 Artificial intelligence

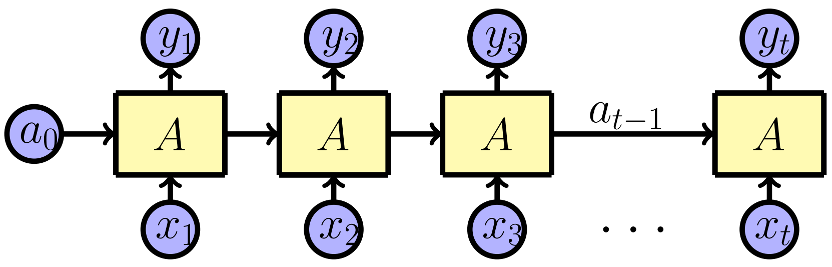
The previously explained concepts belong to what we call 'Artificial Intelligence'. Artificial intelligence is a branch of Informatics with the aim to develop computer systems that are able to perform complicated tasks that usually require human intelligence.

4.1.4 Neural Networks

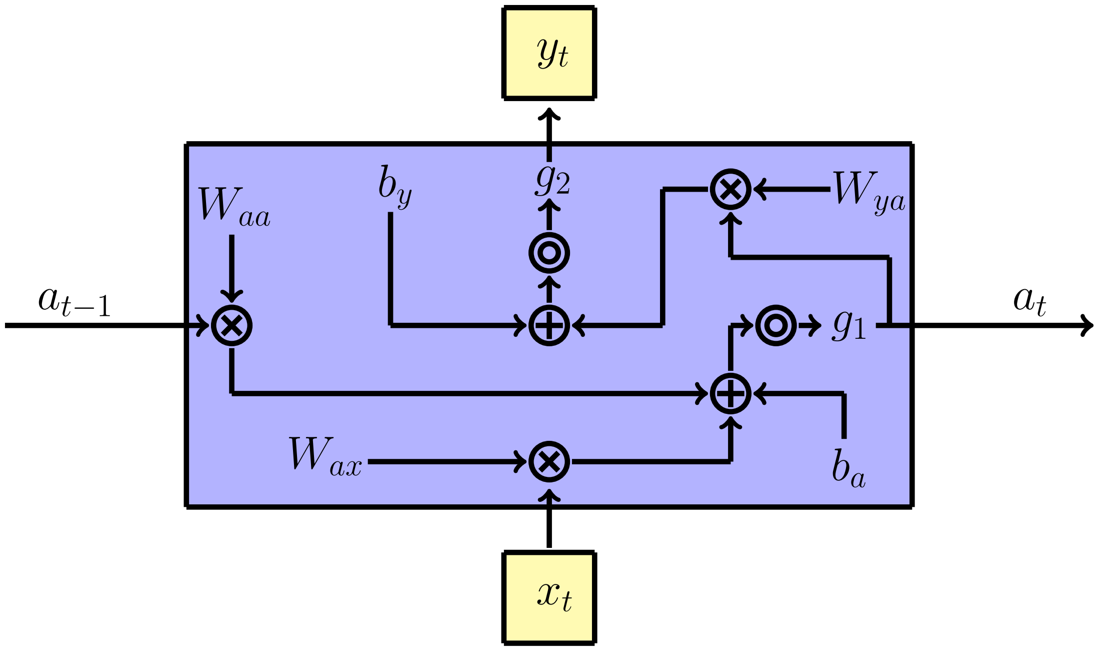
A neural network is a computer system that mimics the human brain to process data. As mentioned above 4.1.2 a neural network is a network with a layered structure where each node is referred to as neuron.

4.2 RNN architecture

Recurrent Neural networks are a type of deep learning algorithms which were initially created in 1980's. They are in fact based on the work of David Rumelhart (1942-2011) who was employed as a Professor of Psychology by the Stanford University, University of California and University



of San Diego.
Recurrent neural networks use aspects of previous outputs when analysing new data inputs to create precise predictions. We're mainly talking about the analysis of financial data but RNNs are used for sequential data like time series, audio, speech, text, weather and many more. Sequential data is ordered data where the order of the data is important since the individual information are related.
Each neuron in the neural network is equipped with a so called activation function that determines the output of the RNN. The activation function decides if a neuron is activated or not while the RNN is running. Depending on whether the neuron's input is relevant to the prediction the activation of the neuron ensues.
The choice of the activation function is fundamental to the accuracy and efficiency of the network. (Should we add the different types and explain them?)



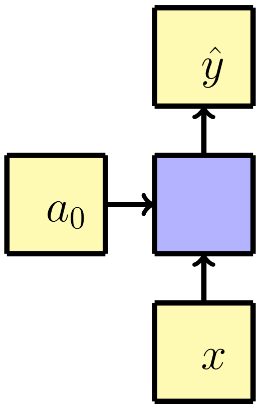
The graph above depicts the architecture of a traditional RNN. For each time step/unit we have the following activation a_t and output y_t:

$$a_t = g_1(W_{aa}a_{t-1} + W_{ax}a_t + b_a)$$
$$y_t = g_2(W_{ya}a_t + b_y)$$

with W_{ax}, W_{aa}, W_{ya}, b_a, b_y being temporarily shared coefficients and g_1 and g_2 being the activation functions.

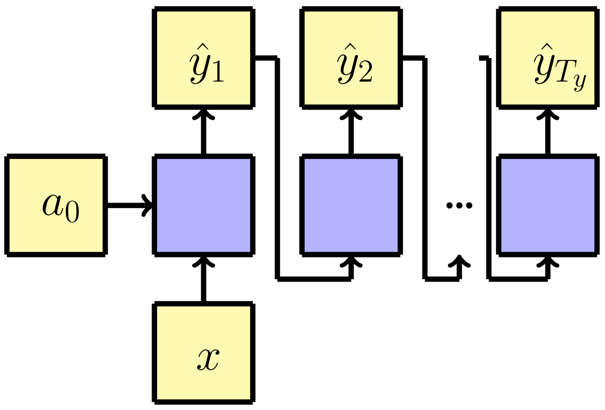
4.3 Types of RNN

One-to-one $T_x = T_y = 1$



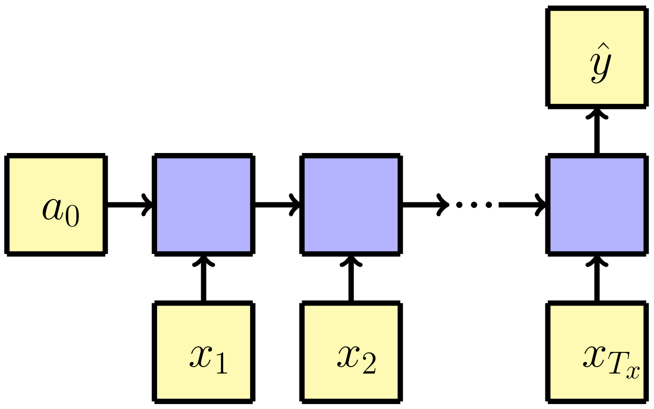
This is the standard neural network where we have one input and one output.

One-to-many $T_x = 1, T_y > 1$



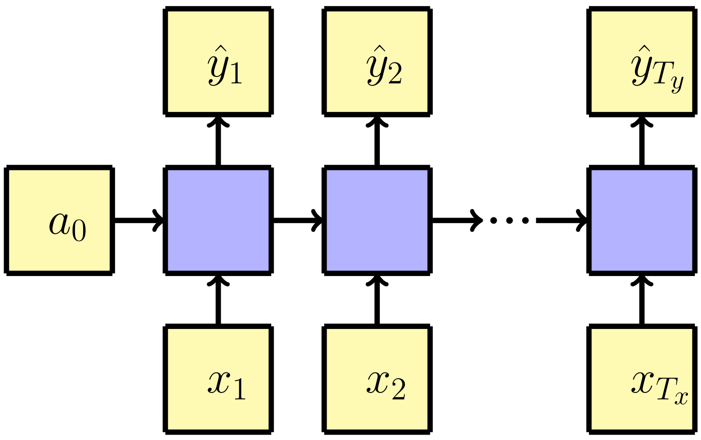
On the diagram above we can see that for a one to many RNN, we use one input at the start and then proceed to use the outputs as the inputs for the following inputs. Hence we get multiple outputs using a single input at the beginning. This type of RNN is used for music generation for example.

Many-to-one $T_x > 1, T_y = 1$



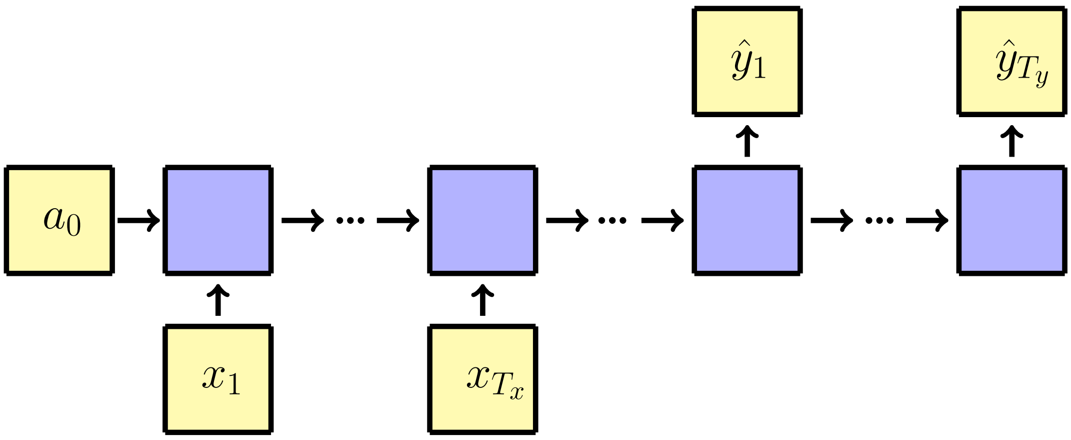
Here we have multiple inputs that are processed by the neural network which yields a single output. An example for this is sentiment classification which is a process that identifies opinions in text and then classifies those. A tangible example would be a set of reviews for a movie that are processed by the neural network which returns the overall rating for the movie.

Many-to-many $T_x = T_y$



The input sequence has the same length as the output sequence. For instance imagine that we have an algorithm that recognizes names in a text and then marks these in the output. The output is essentially the same as the input with the only difference being that the algorithm has marked the names in the next.

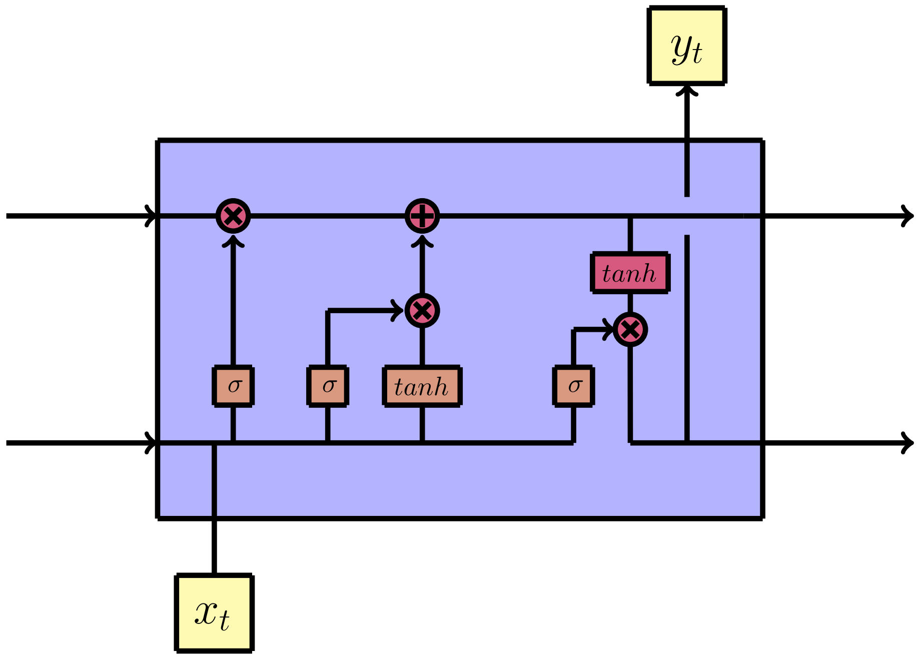
Many-to-many $T_x \neq T_y$



This time the sequence length of the inputs does not have to be the same as the length of the output sequence. In fact by looking at the diagram above we can see that the output sequence starts after the inputs have already been processed by the neural network. An example for this type of RNN would be machine translation. In essence we have a sentence that is split up into multiple inputs and then after processing the whole sentence the neural networks starts returning chunks of the sentence in another language. The output sentence does not necessarily have the same amount of words as the input sentence.

4.4 LSTM

A Long Short-Term Memory network is a type of recurrent neural network.
The traditional RNN layer tends to overwrite a part of its internal memory which is why the neural network is not able to access events further in the past.
Unlike the RNN cell, the LSTM cell doesn't overwrite its internal memory at each step but rather decides depending on the new information if the internal memory is overwritten.
This feature is due to a memory cell that is added in the LSTM architecture. This is also why the LSTM cell architecture is more commonly used compared to the RNN layer architecture.



Taking a look at the LSTM layer diagram, the first thing we see is that instead of having one layer like a RNN cell, there are four layers.
These layers are represented by orange boxes in the lower half of the diagram. The arrows represent vector transfers and the circles and boxes marked in red are point wise operations.
When to arrows come together they represent a concatenation of vectors and if a arrow is divided into two arrows we create a copy of the vector.
The upper line in the diagram is the cell state which is the main current of data. The cell state is then accessed by gates that regulate the addition or removal of data from the cell state.
The gates are structures composed of a sigmoid layer and a point wise operation. The output of the sigmoid layer is between 0 and 1 included. The exact number denotes the amount of information that passes the gate.
It ranges from 0 which means that no information passes, to 1 which means that all the information passes. There are 3 gates in total that regulate the data exchange of the cell state.

4.5 Types of activation functions

Let us take a closer look at some activation functions.

4.5.1 Binary Step Function

The Binary Step function is fairly simple. The function chooses a threshold and if the input is above the threshold then the function output is 1 which activates the neuron and if it is below the threshold then the output is 0 and the neuron doesn't fire off a signal. While the function is simple it also has a fair bit of problems that other functions can handle. For example the Binary step function does not support outputs with multiple values.

4.5.2 Linear Activation Function

The linear activation function creates outputs that are proportional to the inputs. The inputs are multiplied by the weight of the neuron and as a consequence can handle outputs with multiple values unlike the binary step function.
Backward propagation of errors or "Back propagation" is an algorithm that calculates the gradient of the error function while considering the weights of the neural network.
The algorithm starts by calculating the gradient of the last layer before going backwards to the first layer.
The gradient is a vector which tells us in which direction the loss function has the steepest ascent. Ideally the model moves in the opposite direction to decrease the loss function as much as possible. This process is called Gradient Descent. The following activation functions will be non-linear which allows them to use back propagation and stacking multiple layers of neurons to create neural networks. In other words, these will be the functions that we're mainly going to use and compare later on.

4.5.3 Sigmoid

The sigmoid function will be used a later on for the description of the LSTM architecture. The outputs of the sigmoid function lie between 0 and 1 so we have a normalized output. Another advantage is that the output values are smooth in the sense that there is no jump between different values.

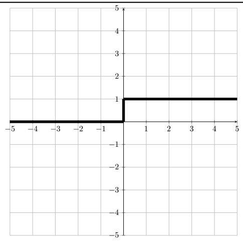
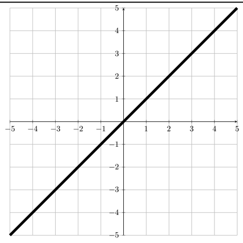
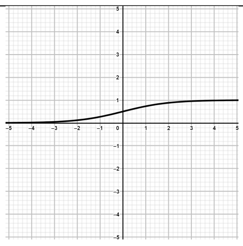
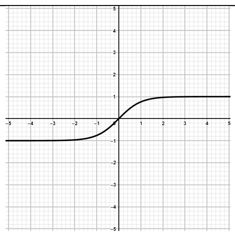
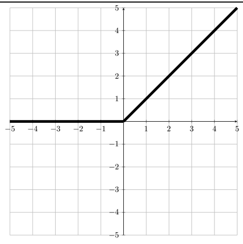
4.5.4 Hyperbolic Tangent

The tanh function is similar to the sigmoid function with the only difference being that it is zero centred.
This means that the tanh function handles inputs that are comprised of strongly negative, neutral and strongly positive values better by normalizing the values in a range around 0.

4.5.5 Rectified Linear Unit (ReLU)

The rectified linear unit may seem linear at first look but it is actually non-linear which gives rise to advantages that linear functions do not possess. For example by having a derivative function the ReLU allows back propagation. The disadvantage to using ReLu is that the network has trouble handling values that are close to zero or negative.
In fact by learning a large negative bias for the neuron weights the ReLu is rendered useless because the outputs will always be 0 if the ReLU ends up in this state. This disadvantage is referred to as the Dying ReLU Problem.

4.5.6 Table of activation functions

Type of activation function	Associated graph	Associated function
Binary step function		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Linear activation function		$f(x) = w \cdot x$ (w=weight constant)
Sigmoid activation function		$f(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent activation function		$f(x)=tanh(x)=\frac{2}{1+e^{-2x}}-1$
ReLU (Rectified Linear Unit) Activation Function		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

5 Experimentation

5.1 Regression line

The observations and experimentation are being treated in section 2 and 6.1.

5.2 Simplified RNN

This simplified code uses a built-in function to split the data set into training and test data. This function also randomizes the data which shouldn't be a problem since we're interested in the precision of the model and not the actual prediction of the stock price.
This code allows us to change the parameters of the model which includes number of layers, type of cells, activation functions,...
Thus we can easier play around with the parameters to see how they affect the model.

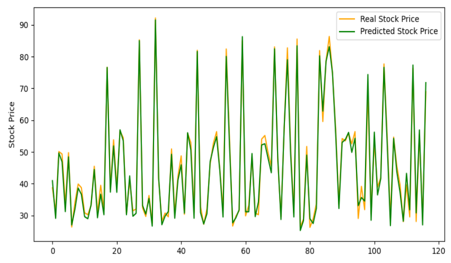


Figure 3

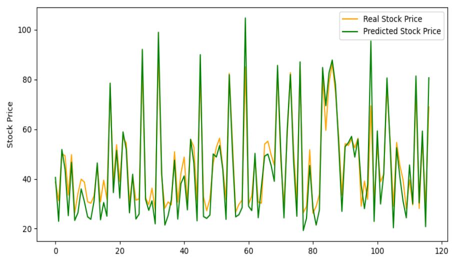


Figure 4

For *Figure 3* and *Figure 4* we used models composed of two LSTM layers with 200 neurons for each of them and one dense layer. Both LSTM layers use ReLU as activation function. The models are trained for 50 epochs with the only difference being that the model for the *Figure 3* uses a batch size of 20 and the model for *Figure 4* uses a batch size of 50. The resulting difference in the two figures is not big but we can see that the model that used a batch size of 60 is not as precise as the model that used a batch size of 20.

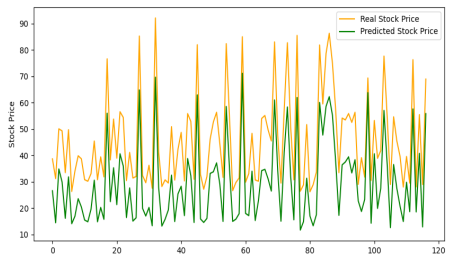


Figure 5

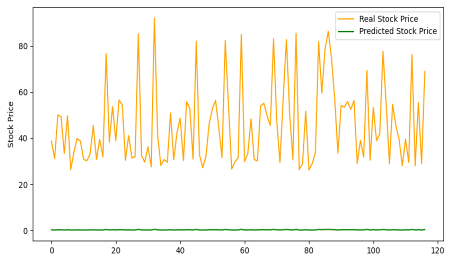


Figure 6

The model for *Figure 5* uses a single LSTM layer with only 50 neurons and as we can see the prediction is already off. For *Figure 6* we go back to using two LSTM layers with 200 neurons. This time we reduced the amount of epochs to 5. As we can see on the *Figure 6* 5 epochs are not enough to train the model and the predicted values are on a straight line.

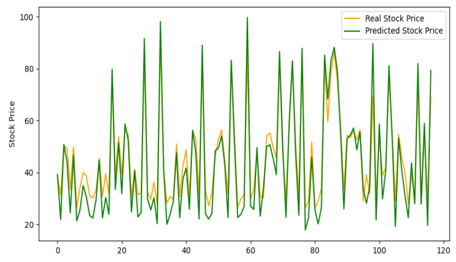


Figure 7

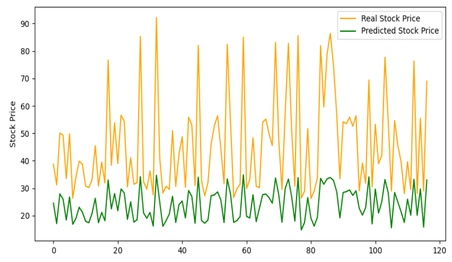


Figure 8

The model used for *Figure 7* is composed of 2 LSTM layers, each having 50 neurons. Once again we use ReLU activation functions, but this time we have a batch size of 50 and we train the model for 200 epochs. Then for *Figure 8* we switched to using a single LSTM layer with 50 neurons followed by a dense layer. The activation function used for the LSTM layer was *tanh* while the batch size was set to 20 and the model was trained for 50 epochs. So we have a model comparable to that of *Figure 5* with the difference being that we used *tanh* as activation function. This resulted in a slightly worse prediction.

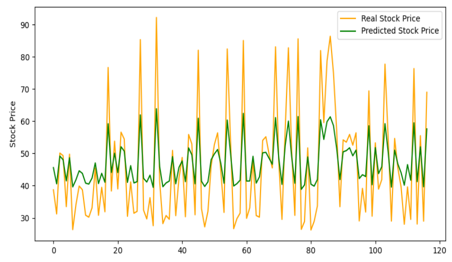


Figure 9

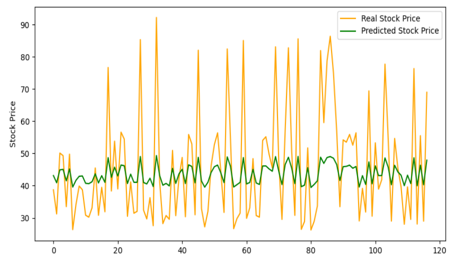


Figure 10

For the figure *Figure 9* we switched the activation function to sigmoid. This shifted the predicted values upwards.
Then for the next model we switched back to using two LSTM layers with 200 neurons each. The model was trained for 100 epochs which strangely enough resulted in a less precise prediction even though we increased the amount of layers and neurons.

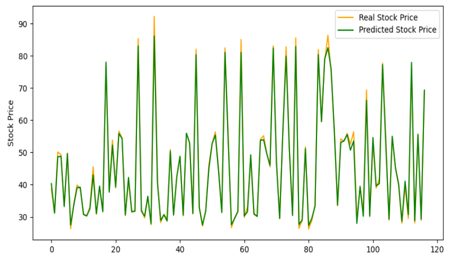


Figure 11

Hence we tried to increase the amount of neurons to 400 for both layers of the model for *Figure 11.* The model was trained for 500 epochs to ensure a precise prediction. In fact we've observed that the loss function has reached an optimal value after 200 epochs and stayed around that value for the remaining 300 epochs.

5.3 Standard RNN and LSTM

Remark : The following table of parameters will be used to plot the figures for the standard RNN and LSTM. Also note that the figures on the left side are created with a RNN model and the figures on the right side are created with a LSTM model.
The first line of the table shows the standard parameters of the program.

figure	valid set size percentage	test set size percentage	seq len	neurons	layers	learningrate	batchsize	epochs	activation
12,13	10	10	20	200	2	0.001	50	100	ReLU
14,15	10	10	20	200	2	0.001	50	100	tanh
16,17	10	10	20	200	2	0.001	50	100	sigmoid
18,19	50	10	20	200	2	0.001	50	100	ReLU
20,21	10	50	20	200	2	0.001	50	100	ReLU
22,23	10	10	50	200	2	0.001	50	100	ReLU
24,25	10	10	20	50	2	0.001	50	100	ReLU
26,27	10	10	20	200	1	0.001	50	100	ReLU
28,29	10	10	20	200	2	0.001	20	100	ReLU
30,31	10	10	20	200	2	0.001	100	100	ReLU

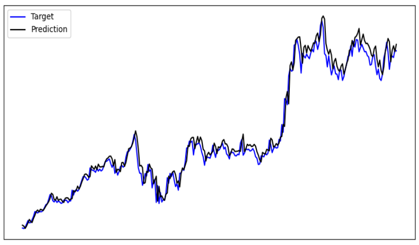


Figure 12: Standard parameters

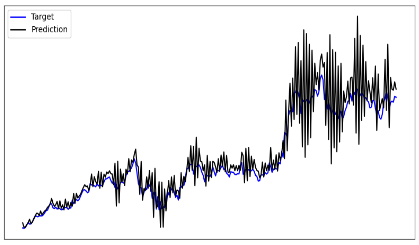


Figure 13: Standard parameters

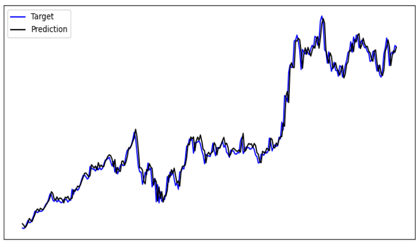


Figure 14: tanh instead of ReLU

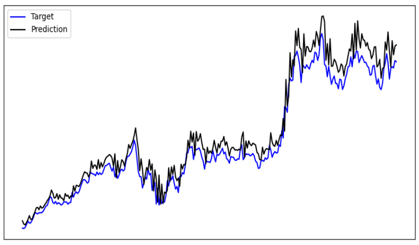


Figure 15: tanh instead of ReLU

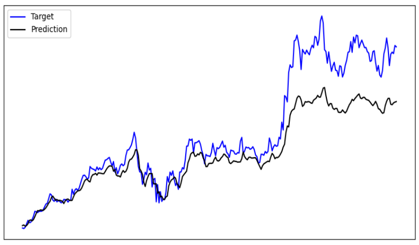


Figure 16: sigmoid instead of ReLU

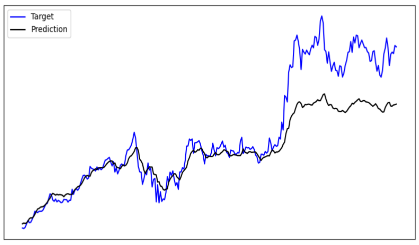


Figure 17: sigmoid instead of ReLU

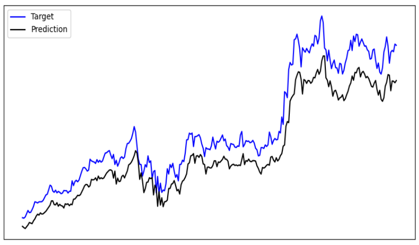


Figure 18: increased valid set size percentage

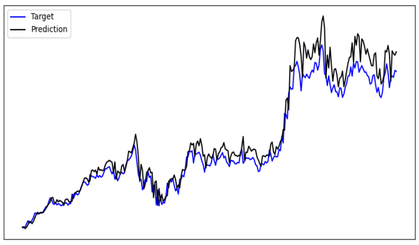


Figure 19: increased valid set size percentage

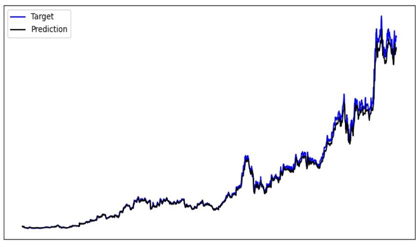


Figure 20: increased test set size percentage

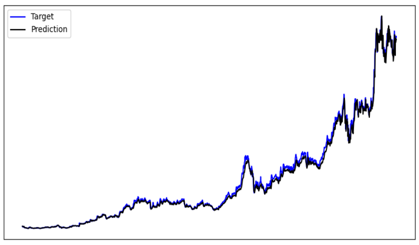


Figure 21: increased test set size percentage

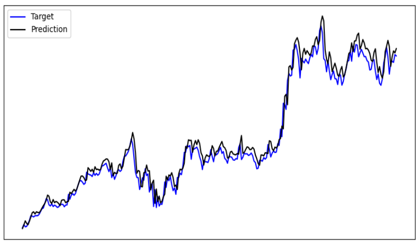


Figure 22: increased sequence length

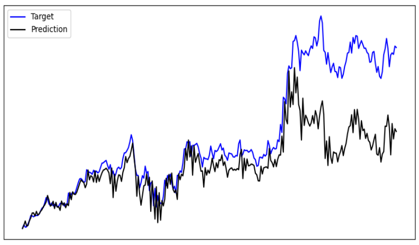


Figure 23: increased sequence length

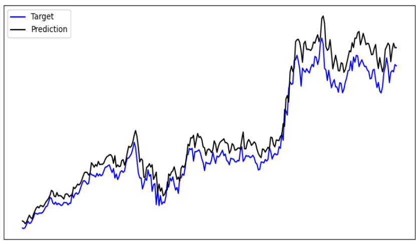


Figure 24: 50 instead of 200 neurons

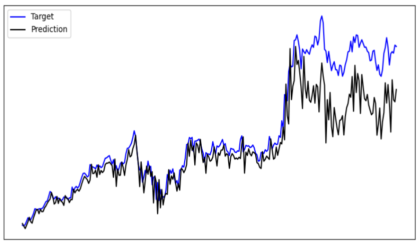


Figure 25: 50 instead of 200 neurons

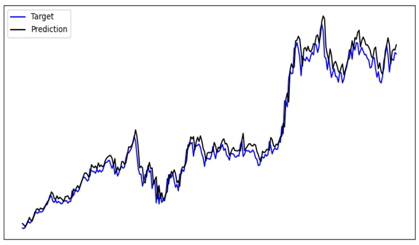


Figure 26: 1 instead of 2 layers

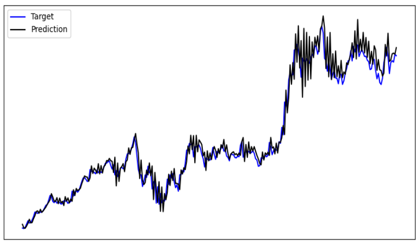


Figure 27: 1 instead of 2 layers

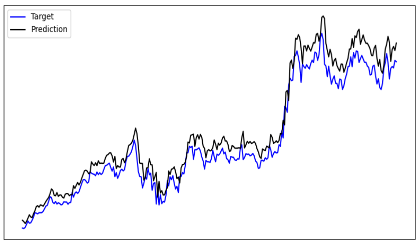


Figure 28: batchsize 20 instead of 50

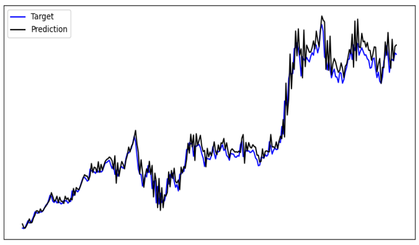


Figure 29: batchsize 20 instead of 50

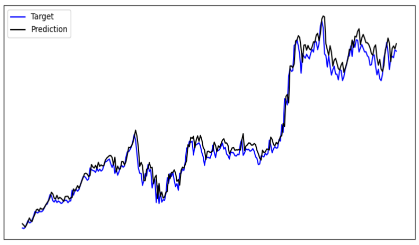


Figure 30: batchsize 100 instead of 50

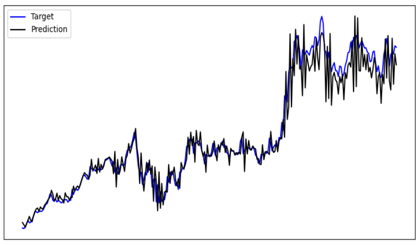


Figure 31: batchsize 100 instead of 50

6 Python code

6.1 Regression line with error-bars and R^2-coefficient

```python
# Imports that we need for this program

import matplotlib.pyplot
from matplotlib import style
import pandas_datareader.data as pdt
import datetime as dt
import math
import numpy as np

# Getting the data we need

tst = dt.datetime(2019, 1, 1)
ted = dt.datetime(2019, 12, 31)

data1 = pdt.DataReader('AMD', 'yahoo', tst, ted)

closelist = data1['Close'].tolist()

# Method of linear regression

N = len(closelist)

days = []
for i in range(len(closelist)):
    days.append(i)

Xbar = sum(days)/N

Ybar = sum(closelist)/N

Znum = []
for i in range(len(closelist)):
    Znum.append(days[i]*closelist[i])
o = sum(Znum)

Zden = []
for i in range(len(closelist)):
    Zden.append(days[i] * days[i])
p = sum(Zden)

alpha = ((o / N) - (Xbar * Ybar)) / ((p/N) - (Xbar * Xbar))
beta = Ybar - alpha * Xbar

regline = []
for i in range(len(closelist)):
    regline.append(alpha * i + beta)

# R^2 coefficient

y_isqr = []
for i in range(len(closelist)):
    y_isqr.append(closelist[i] * closelist[i])
yy = sum(y_isqr)

sigma_xy = (((o/N) - (Xbar * Ybar)) / (math.sqrt((p/N) - (Xbar * Xbar)) * math.
    sqrt((yy/N) - (Ybar * Ybar))))**2

# Error-bars for linear regression

y_hat_i = []
for i in range(len(days)):
    y_hat_i.append(alpha * i + beta)

e_i = []
for i, j in zip(y_hat_i, closelist):
    e_i.append(j - i)

# Plotting the data and return the R^2 coefficient

```

```python
69 print('R^2=', sigma_xy)
70
71 style.use('seaborn')
72 matplotlib.pyplot.plot(regline)
73 matplotlib.pyplot.errorbar(days, closelist, yerr=e_i, fmt='k')
74 matplotlib.pyplot.show()
```

6.1.1 Explanation of the code

- Line 3-8 : Here we simply import the libraries we need for the program
- Line 12-17 :
  - For 'tst' and 'ted' one can choose the time period for the linear regression
  - 'data1' gives the data of AMD from the yahoo finance database
  - We create a list called 'closelist' containing the closing price of the AMD stock
- Line 21-46 :
  - In line 21 we define 'N' as the length of the closelist and in lines 23-25 we create a list containing integers from 1 to n, where n is the length of the closelist (the length of the list is not equal to the length of the time period we defined for 'tst' and 'ted', because of weekends and holidays)
  - In line 27 and line 29, 'Xbar' respectively 'Ybar' represent the formula seen for $\bar{x}$ and $\bar{y}$ in section 2.2
  - In lines 31-34 we determine $\sum_{i=min\{N\}}^{max\{N\}} x_i y_i$ and in lines 36-39 $\sum_{i=min\{N\}}^{max\{N\}} x_i^2$ (see section 2)
  - In line 41 we calculate α (see section 2.6) and in line 42 β (see section 2.6)
  - In line 44-46 we calculate the regression line (see section 2.6)
- Line 50-55 :
  - In lines 50-53, we determine $\sum_{i=min\{N\}}^{max\{N\}} y_i^2$
  - In line 55, we calculate σ_{xy} seen in section 2.2 (Note: $R^2 = \rho_{xy}^2$)
- Line 59-65 :
  - In lines 59-61 we determine $\hat{y}_i$ (see section 2.4)
  - In lines 63-65 we determine the e_i (see section 2.4)
- Lines 71-74 :
  - In line 71 we choose a certain style for the output
  - In line 72 we plot the regression line calculated in the 'Method of linear regression'
  - In line 73 we plot the error-bars (e_i) for every point (x_i, y_i) in the cloud of points
  - In line 74 we lastly give the command to show the plot

6.2 Simplified RNN

```python
# import libraries needed for the program

import pandas
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
from sklearn.model_selection import train_test_split

```

```

# load data

data = pandas.read_csv('AMD.csv')

# dropping date column and changing order of the columns

data = data[['close', 'volume', 'open', 'high', 'low']]

# iloc uses the index to choose columns

data_train = data.iloc[:, 1:]
target_values = data.iloc[:, 0]

# return data frame as numpy array

data_train = data_train.values

# Normalizing data set

scaler = MinMaxScaler(feature_range=(0, 1))
data_train = scaler.fit_transform(data_train)
target_values = target_values.values

# splitting data set in train and test data

X_train, X_test, y_train, y_test = train_test_split(data_train, target_values,
    test_size=0.33, random_state=42)

# changing the input shape for the model

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

# creating model and adding the different layers

layers = Sequential()

layers.add(LSTM(200, activation='relu', return_sequences=True))
layers.add(Dropout(0.2))

layers.add(LSTM(200, activation='relu'))
layers.add(Dropout(0.2))

layers.add(Dense(1))

# Compiling the model

layers.compile(loss='mean_squared_error', optimizer='adam')

# training the model using the training data

layers.fit(X_train, y_train, epochs=50, batch_size=20, verbose=1)

# predicting the stock value

out_predict = layers.predict(X_test)

# Visualisation of the stock prices and estimated values

plt.style.use('dark_background')
plt.figure(figsize=(10, 5))
```

```
72 plt.plot(y_test , color='orange', label='Real Stock Price')
73 plt.plot(out_predict , color='green', label='Predicted Stock Price')
74 plt.ylabel('Stock Price')
75 plt.legend()
76 plt.show()
```

6.2.1 Explanation of the code

- Line 3-8 : We start by importing the libraries needed for the code
- Line 14 : We use pandas to create a data frame by loading a data set from a .csv file In our case we use a .csv file containing the stock of AMD from 2019-05-02 to 2020-09-23
- Line 18 : We proceed by dropping the date line and rearranging the data set
- Line 22-23 :
    - Line 22 : We take every column except the 'close' column
    - Line 23 : We define target_place as the 'close' column
- Line 27 : The data frame is converted into a numpy array
- Line 31-33 :
    - Line 31 : The scaler is defined as MinMaxScaler; MinMaxScaler is a function imported from the sklearn.preprocessing library that allows us to normalize the data set with values between 0 and 1. In fact the function compares the greatest element in the data with the smallest to create a scale.
    - Line 32 : MinMaxScaler is applied to our data set
    - Line 33 : The list with close values is converted into a numpy array
- Line 37 :
        - The data set is split into train and test data using the imported function train_test_split()
        - The function input corresponds to our data set which is split into data_train and target_values
        - test_size defines the portion of data that is taken as test data
        - The data is randomized by the function and we use random_state to choose how the data is randomized. In other words we can achieve the same random data set by using the same random_state. This allows us to test different parameters of our model without getting different results due to the randomization
- Line 41-42 : The x_train and x_test data which is in the form of a numpy array is converted to a form that the model can use
- Line 46-54 :
    - The model is chosen as a Sequential model using the Sequential() function from the Keras.models library
    - Line 48 : We add a LSTM layer with 200 neurons and choose an activation functions
    - Line 49 : The dropout layer prevents over fitting of the model by setting random neuron values to 0 The dropout is only applied during the training. Otherwise we would have a loss of information concerning our prediction
- Line 58 : The model is compiled and we choose the loss function and optimizer
- Line 62 :
    - The model is trained using the training data and the following parameters as input:
        - epochs defines the amount of iterations for the training data
        - batch_size corresponds to the size of data the model uses at each step for finding patterns
        - verbose defines the program output;
          If it is 0 we don't see the training procedure in our output
          For verbose=1 we get a beam showing the progress
          verbose=2 only shows the amount of epochs
- Line 70-76 : The matplotlib library allows us to represent the test values compared to the estimated values

6.3 Standard RNN and LSTM

```python
# import all libraries

import numpy as np
import pandas as pd
import sklearn
import sklearn.preprocessing
import datetime
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.framework import ops
import pandas_datareader.data as pdt

# Time frame

tst = datetime.datetime(2010, 1, 1)
ted = datetime.datetime(2020, 11, 18)

# import dataset

dataset = pdt.DataReader('AMD', 'yahoo', tst, ted)
df_stock = dataset.copy()
df_stock = df_stock.dropna()
df_stock = df_stock[['Open', 'High', 'Low', 'Close']]

# data scaling (normalizing)

def normalize_data(df):
    min_max_scaler = sklearn.preprocessing.MinMaxScaler()
    df['Open'] = min_max_scaler.fit_transform(df.Open.values.reshape(-1, 1))
    df['High'] = min_max_scaler.fit_transform(df.High.values.reshape(-1, 1))
    df['Low'] = min_max_scaler.fit_transform(df.Low.values.reshape(-1, 1))
    df['Close'] = min_max_scaler.fit_transform(df['Close'].values.reshape(-1, 1))
    return df

df_stock_norm = df_stock.copy()
df_stock_norm = normalize_data(df_stock_norm)

# Splitting the dataset into Train, Valid & test data

valid_set_size_percentage = 75
test_set_size_percentage = 10
seq_len = 20  # taken sequence length as 20

def load_data(stock, seq_leng):
    data_raw = stock.to_numpy()  # stock.as_matrix()
    data = []
    for index in range(len(data_raw) - seq_leng):
        data.append(data_raw[index: index + seq_leng])
    data = np.array(data)
    valid_set_size = int(np.round(valid_set_size_percentage / 100 * data.shape[0])
    )
    testt_set_size = int(np.round(test_set_size_percentage / 100 * data.shape[0]))
    ttrain_set_size = int(data.shape[0] - (valid_set_size + testt_set_size))
    x_ttrain = data[:ttrain_set_size, :-1, :]
    y_ttrain = data[:ttrain_set_size, -1, :]
    x_validt = data[ttrain_set_size:ttrain_set_size + valid_set_size, :-1, :]
    y_validt = data[ttrain_set_size:ttrain_set_size + valid_set_size, -1, :]
    x_ttest = data[ttrain_set_size + valid_set_size:, :-1, :]
    y_ttest = data[ttrain_set_size + valid_set_size:, -1, :]
    return [x_ttrain, y_ttrain, x_validt, y_validt, x_ttest, y_ttest]
```

```python

x_train, y_train, x_valid, y_valid, x_test, y_test = load_data(df_stock_norm,
    seq_len)

"""Building the Model"""

# parameters & Placeholders

n_steps = seq_len - 1
n_inputs = 4
n_neurons = 200
n_outputs = 4
n_layers = 2
learning_rate = 0.001
batch_size = 50
n_epochs = 50
train_set_size = x_train.shape[0]
test_set_size = x_test.shape[0]
ops.reset_default_graph()
X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_outputs])

# function to get the next batch

index_in_epoch = 0
perm_array = np.arange(x_train.shape[0])
np.random.shuffle(perm_array)

def get_next_batch(batch_sizee):
    global index_in_epoch, x_train, perm_array
    start = index_in_epoch
    index_in_epoch += batch_sizee
    if index_in_epoch > x_train.shape[0]:
        np.random.shuffle(perm_array)  # shuffle permutation array
        start = 0  # start next epoch
        index_in_epoch = batch_sizee
    end = index_in_epoch
    return x_train[perm_array[start:end]], y_train[perm_array[start:end]]

# RNN
layers = [tf.contrib.rnn.BasicRNNCell(num_units=n_neurons, activation=tf.nn.relu)
          for layer in range(n_layers)]
# LSTM
# layers = [tf.contrib.rnn.BasicLSTMCell(num_units=n_neurons, activation=tf.nn.
    relu)
#           for layer in range(n_layers)]

multi_layer_cell = tf.contrib.rnn.MultiRNNCell(layers)
rnn_outputs, states = tf.nn.dynamic_rnn(multi_layer_cell, X, dtype=tf.float32)
stacked_rnn_outputs = tf.reshape(rnn_outputs, [-1, n_neurons])
stacked_outputs = tf.layers.dense(stacked_rnn_outputs, n_outputs)
outputs = tf.reshape(stacked_outputs, [-1, n_steps, n_outputs])
outputs = outputs[:, n_steps - 1, :]  # keep only last output of sequence

# Cost function

loss = tf.reduce_mean(tf.square(outputs - y))

# optimizer

optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

```
120
121 # Fitting the model
122
123 with tf.Session() as sess:
124     sess.run(tf.global_variables_initializer())
125     for iteration in range(int(n_epochs * train_set_size / batch_size)):
126         x_batch, y_batch = get_next_batch(batch_size)  # fetch the next training
    batch
127         sess.run(training_op, feed_dict={X: x_batch, y: y_batch})
128         if iteration % int(5 * train_set_size / batch_size) == 0:
129             mse_train = loss.eval(feed_dict={X: x_train, y: y_train})
130             mse_valid = loss.eval(feed_dict={X: x_valid, y: y_valid})
131             print('%.2f epochs: MSE train/valid = %.6f/%.6f' % (
132                 iteration * batch_size / train_set_size, mse_train, mse_valid))
133 # Predictions
134     y_test_pred = sess.run(outputs, feed_dict={X: x_test})
135
136 # plotting the graph
137
138 comp = pd.DataFrame({'Column1': y_test[:, 3], 'Column2': y_test_pred[:, 3]})
139 plt.figure(figsize=(10, 5))
140 plt.plot(comp['Column1'], color='blue', label='Target')
141 plt.plot(comp['Column2'], color='black', label='Prediction')
142 plt.tick_params(left=False, bottom=False, labelleft=False, labelbottom=False)
143 plt.legend()
144 plt.show()
```

Remark: Program by Umesh Palai (Reference: [9])

6.3.1 Explanation of the code

- Lines 3-11 : We import the libraries that we will use for this program
- Lines 15-16 : We determine the time period for which we want to consider data
- Lines 20-23 :
  - Line 20 : We import the data set of AMD for the given time period
  - Lines 21-22 : We use the dropna() function to exclude the missing data values
  - Line 23 : We consider the OHLC columns of the data set
- Lines 27-36 : We use sklearn‘s ‚MinMaxScaler‘ to normalize the data set to turn the mean of all the input features to 0 and their variance to 1. This is necessary to avoid different scales while training the model. Examples: ‚*tanh*‘ is defined on the $[-1, 1]$ interval ; ‚sigmoid‘ is defined on the $[0, 1]$ interval
  Usually we use the ‚ReLu‘ (rectified linear unit) activation and this activation function is unbounded on the axis of possible activation values.
- Lines 40-61 : The data set is split into train, valid and test data and we chose sequences of length 20.
  So we can build $x_{train}, y_{train}, x_{valid}, y_{valid}, x_{test}, y_{test}$.
- Lines 67-79 : Here we fix the parameters, placeholders and variables to build the models.
  The model is fit by setting placeholders. X contains the networks input (OHLC at time $T = t$) and Y the network‘s output (price of the stock at time $T + 1$)
- Lines 83-110 :
  - Lines 83-96 : We define a function that runs the next batch for every model.
  - Lines 99-110 : We define layers for 2 different models (RNN, LSTM) Note: Only one can run at the same time
- Line 114 : The loss function is used to optimize the model. It measures the deviation between the network‘s predictions and the observed training targets.

- Lines 118-119 : The optimizer calculates gradients that indicate the direction in which the prediction has to be changed during the training in order to minimize the loss function.
  We use Adam (Adaptive Moment estimation) here. It is one of the default optimizers in deep learning development.
- Lines 123-134 : Here we fit the model to our training data set. We train the model using batches, where data samples of n = batch_size are fed into the network. The training data set is divided into n/batch_size batches that are sequentially fed into the network. Here the placeholders X and Y play a role, as they store the input and target data and present these to the network.
  The grouped data of X flows through the system until it reaches the output layer. Then tensorflow compares the predictions of the model to the targets Y for the current batch.
  Then after this step, it optimizes and updates the parameters of the network. After the optimization process it considers the next batch and repeats the same process until all the batches have been processed. (One iteration through all the batches is called an epoch. The program stops when the chosen amount of epochs has been reached)
- Lines 138-144 : Here we simply plot the prediction and the target. Note that the target (y_test) and the predicted closing prices (y_test_pred) are put into one data frame called ‚comp'.

7 Conclusion

First we could observe that the linear regression is not the best method to make a prediction as it is obvious that the main factor for variation in a stock is time. In our case it is useful to show a general trend of the stock but it is not a precise prediction tool.

The simplified RNN we described in section 5.2 has the goal of playing around with parameters and to optimize the model to give more precise predictions. As the stock values are chosen at random, it isn't useful to make predictions for investment or to analyse future stock values.

Then for the stock prediction using machine learning we've seen that by taking a look at *Figures 12 - 31* that the ReLU activation function leads to a more precise prediction compared to *tanh* and sigmoid. While the choice of the activation function didn't influence the speed of the prediction, we've observed that the LSTM architecture was slightly slower than the RNN architecture.
From the experimentation we can derive that an increase in the valid set size percentage and the test set size percentage result in a faster output, whereas the increased test set size percentage also increases the range of the output data (Reference: *Figures 20 and 21*).
An increase of the sequence length (seqlen) reduces the speed of the output generation. The decrease of the number of neurons and layers accelerates the rate at which the output is generated but results in a less precise prediction. As for the batch size we found out that a lower batch size improves the precision of the prediction but also increases the time needed to train the model. An increase in the batch size leads to the exact opposite. Last but not least we could observe that the LSTM model has a much bigger deviation from the target compared to the RNN model.

References

[1] 7 Types of Activation Functions in Neural Networks: How to Choose? https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[2] Backpropagation | Brilliant Math & Science Wiki. https://brilliant.org/wiki/backpropagation/.

[3] CS 230 - Recurrent Neural Networks Cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[4] Deep learning vs machine learning. https://www.zendesk.com/blog/machine-learning-and-deep-learning/.

[5] Deep learning vs. machine learning – What's the difference? https://levity.ai/blog/difference-machine-learning-deep-learning.

[6] Different types of RNNs - Recurrent Neural Networks. https://www.coursera.org/lecture/nlp-sequence-models/different-types-of-rnns-BO8PS.

[7] Understanding LSTM Networks – colah's blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[8] Intro to optimization in deep learning: Gradient Descent. https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/, June 2018.

[9] RNN, LSTM, And GRU For Trading. https://blog.quantinsti.com/rnn-lstm-gru-trading/, December 2018.

[10] Discrete time and continuous time. *Wikipedia*, October 2020.

[11] eToro's Users Soar After The Platform Went Commission-Free On Stock Trading, January 2020.

[12] Markov decision process. *Wikipedia*, November 2020.

[13] James Chen. Neural Network Definition. https://www.investopedia.com/terms/n/neuralnetwork.asp.

[14] Jerry Magar CNP-Sciences économiques et sociales (ES). *Élaboration d'un cours de «Statistique et Probabilités» pour la classe de 1D.*

[15] deeplizard. Markov Decision Processes (MDPs) - Structuring a Reinforcement Learning Problem, September 2018.

[16] The MIT Press. David E. Rumelhart. https://mitpress.mit.edu/contributors/david-e-rumelhart.

[17] SAGAR SHARMA. Activation Functions in Neural Networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, February 2019.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[19] Expert System Team. What is Machine Learning? A definition, May 2020.

[20] Avinash Sharma V. Understanding Activation Functions in Neural Networks. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, March 2017.