A Machine Learning Based DEVS Simulator to Predict Stock Prices

|  |  |
| --- | --- |
| Guillermo G. Trabes | Gabriel A. Wainer |
| Department of Systems and Computer  Engineering, Carleton University and  Universidad Nacional de San Luis  1125 Colonel By Dr.  Ottawa, ON K1S 5B6, CANADA | Department of Systems and Computer Engineering, Carleton University  1125 Colonel By Dr.  Ottawa, ON K1S 5B6, CANADA  gwainer@sce.carleton.ca |
| guillermotrabes@sce.carleton.ca |  |
|  |  |
| Veronica Gil Costa | |
| Universidad Nacional de San Luis  And CCT CONICET  Ejercito de Los Andes 950  San Luis, D5700, Argentina  gvcosta@unsl.edu.ar | |

ABSTRACT

Stock price prediction is one of the most interesting problems in finance. Thousands of stocks are traded every day thought the world and their prices fluctuations are followed by the markets. Any additional information about the future can have a significative impact on the decisions investors made. To achieve this goal, several efforts have been made, one of the most important advances in this field is the use of artificial intelligence to predict future prices. Nevertheless, few attempts have been made to incorporate modeling and simulation techniques that allow to evaluate the quality of the decisions made by predictors and the actual return they can achieve. In this work a machine learning DEVS based simulator is presented that allows to simulate and evaluate the behavior of a decision support system for stock trading.

**Keywords:** DEVS, Stock price prediction, machine learning, deep learning.

# INTRODUCTION

Decision-making in stock trading system is an emerging research area and has great commercial potential. In stock market, investors buy stocks in the hope that it will yield income from dividends and appreciate, or grow, in value. The uncertain movements or random behavior of the stock price is due to the concept of efficient market hypothesis (EMH) i.e., past stock prices will influence the present stock values and the markets will respond quickly to any new information related to a stock. In other words, the price of a stock at any given point of time is a product of both its history of price levels and its new value. Therefore, the expected future value of a stock depends only on its current price and past stock prices. Stock market forecasters focus on predicting the values of indexes and individual stocks, and to give a recommendation to the investor: buy, sell or hold the stock.

In the past few years several stock prices predictors have been made, mainly with the development of advance machine learning techniques, as deep learning. Deep learning is a technique that uses neural networks capable of learning unsupervised from data that is unstructured or unlabeled.

The approach proposed in this work combines the best of both words. On one hand, the use of a deep learning to predict the stocks’ future prices, which is very difficult to model by traditional M&S (Modeling and Simulation) techniques and, on the other hand uses a well-known formalism for M&S, DEVS, to define a model that gives a decision support system to the investors. Also DEVS is used to simulate the use of the decision support system and to predict the error on the predictions and the decisions suggested to the investors.

An implementation of the proposed model was made using the TensorFlow framework for Deep Learning and the Cadmium tool to define the DEVS model. Experiments were conducted to show how the accuracy of the decision support system, and finally, the conclusions are presented.

# BACKGROUND

In this section, the basic concepts used in this work are presented. In first place the general ideas from Machine Learning are presented, then the DEVS model is detailed and finally the previous attempts to combine M&S with machine learning are discussed.

## Machine Learning

Machine learning is a branch of artificial intelligence (AI) that provides models the ability to automatically learn and improve from experience without being explicitly programmed. It focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples or direct experience, in order to look for patterns in data and make better decisions in the future based on the examples that provided. The primary aim is to allow the computers learn automatically without or little human intervention or assistance and adjust actions accordingly.

One of the most promising ideas within machine learning techniques is Deep Learning. Neural networks are the key components in Deep Learning, they are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a [labeled dataset to train on](https://pathmind.com/wiki/supervised-learning). With classification, Deep Learning is able to establish correlations between: pixels in an image and the name of a person. You might call this a static prediction.

The following figure shows an example of a Deep Learning neural network:

A picture containing building

Description automatically generated

Each connection between neurons is associated with a weight. This weight dictates the importance of the input value. Every neuron calculates a linear function of the inputs and the weights, and then a non-linear function is applied with the purpose of allowing the network to represent non linear functions.

The initial weights are set randomly. A function that shows how wrong the outputs are from the real outputs is defined. This function is called the Cost Function. When the models start to learn from examples the weights are changed according to an optimization method based the cost function’s negative gradient. This way the neural network learns the weights to approximates the expected output values from the inputs.

One key factor of Deep Learning, is that exposed to enough of the right data, the neural network is able to establish correlations between present events and future events. It can run regression between the past and the future. The future event is like the label in a sense. Deep learning doesn’t necessarily care about time, or the fact that something hasn’t happened yet. Given a time series, deep learning may read a string of number and predict the number most likely to occur next. In this work this technique is used to predict from a time series prices of a stock to predict the next value.

## DEVS

DEVS is the formalism proposed by Zeigler in the 70s. This formalism has been implemented in multiple languages and platforms. In DEVS, the modeling hierarchy has two kinds of components: atomic models and coupled models. The atomic models are defined as a tuple: A = <*S, X, Y, int,ext, ta*> where: *S* is the set of states, *X* is the set of input ports and values, *Y* is the set of output ports and values, *int*: *S* → *S* is the internal transition function, *Q* = {(s, e) | s ∈ *S*, 0 ≤ e ≤ *ta*(s)} is the local state set (where e is the time elapsed since last transition), *ext*: *Q* × *X* → *S* is the external transition function, *S* → *Y* is the output function, and *ta* : *S* → ℝ+ is the time-advance function.

An atomic model, also known as a basic model, is always in a specific state waiting to complete the lifespan delay returned by the *ta* function, unless an input of a new external event occurs. If no external event is received during the lifespan delay, the output function ** is called first, and then the state is changed according to the value returned by the *int* function. If an external event is received, then the state is changed according to the value returned by the *ext* function, but no output is generated.

Coupled models define a network structure in which nodes are any Classical-DEVS models (coupled or basic) and oriented links represent the routing of events between outputs and inputs or to/from upper level. Formally, a Coupled Model is represented by the tuple C = <*X*, *Y*, *D*, *M*, *Cxx,* *Cyx,Cyy, SELECT>*,where: *X* is the set of input events, *Y* is the set of output events, *D* is an index for the Classical-DEVS component models of the coupled model, *M* = {Md | d ∈ *D* } is a tuple of Classical-DEVS models as previously defined, *Cxx* ⊆ ∪i ∈ *D* (*Yi*) × ∪i ∈ *D* (*Xi*) is the set of the internal couplings, *Cyy* :∪i ∈ *D* (*Yi*) × *Y* is the external output coupling function, *SELECT:* 2D \ ∅ → *D* is the tie-breaker function that sets priority in case of simultaneous events.

The main advantages of DEVS are the ability to formally defined models, the hierarchical structure that allows to define modular components that can be exchange and the explicit separation between the modeling and the simulation. For these reasons, the use of DEVS in this work will provide benefits in the simulation development.

## Machine Learning Based M&S

As recently stated in (Antoine-Santoni 2019) researches in the field of Modeling and Simulation (M&S) have intensively evolved towards hybrid approaches combining M&S background and Artificial Intelligence advances in data mining and machine learning. The next generation of Modeling & Simulation applications will integrate big data and deep learning tools and methods: “bringing all the three topics together will create synergy that will allow us to significantly improve our services to others science”.

According to literature in the field of M&S, one of the main benefits of the use of Machine Learning is to improve the efficiency simulation analysis. In other words, it may help to reduce simulation cost. In (Saadawi, 2016), it has been proposed to use machine learning mechanisms inside a DEVS simulator in order to optimize simulation execution by learning from past simulations.

A different approach in this field have been made in (Mahmoud, 2018). In this work, the machine learning techniques are used to help the modeling process, part of the model is defined by machine learning components. The drawback of this work is that it lacks a formal definition for the model, as can be defined by DEVS. The proposal of our work follows this idea, to design a simulation that includes a machine learning algorithm in order to predict future stock prices. In addition, the definition of the model with the DEVS formalism allow to reuse components and the formal definition can be implemented using different implementation tools.

This work combines the best of both worlds, the pattern recognition that can be achieve through Deep Learning, and hard to model by hand, with the formal specification and the modeling and simulation capabilities provided by DEVS. To the best of our knowledge, this is the first work using this approach to design a decision support system for stock trading.

# PROPOSED MODEL

The models proposed is a DEVS implementation of a support decision system based on Machine Learning techniques. The model proposed is based on Deep Learning, therefore, a neural network needs to be defined and trained to use as the prediction component.

The model is a 5-level neural network with 60 inputs and 1 output. The 60 inputs are the 60 past values of a stock and the output is the predicted value. The neural network is implemented with the framework TensorFlow and using the Keras library. These libraries execute over the Python language.

All the files needed to train the model are inside the folder ***machine\_learning.*** The ***trainer.py*** file contains the model’s implementation. The execution of this file gives as a result 2 files: ***trained\_model.h***, which is the trained model and the file ***scaler.save***, this file contains the internal scale factor needed by the neural network to execute. The scaler file is necessary cause TensorFlow scales the values from the test set to values between 0 and 1.

The trained model is used as a tool inside the Stock Price Prediction DEVS model that has the following structure and components:

ACCURACY

EVALUATOR

PREDICTION

GENERATOR

In the model’s definition a datatype is used, called time\_series\_element, which is composed by a tuple: <date, price>, where: date is a string representing a correct date on the Gregorian calendar with the format “YYYY-MM-DD”, and price is a real number.

The Prediction Generator model receives as an input 61 time\_series\_element values from a stock and returns 3 time\_series\_element values: the more recent past price from the stock, the real value for the stock and the predicted value for the stock on a particular day.

The accuracy evaluator receives the 3 values given by the prediction generator module and returns 2 metrics, the average estimation error between the real and predicted values, and the decision accuracy made by the model. The average estimation error tracks the average error that the models makes as it predicts prices for different days through time. The decision accuracy is obtained by comparing accuracy of a decision support system given to the investor, buy if the predicted value is less than the most recent price, sell if the predicted value is higher than the last price, and hold if the predicted value is equal to the last price. The values are compared to the ones obtained with the real value to calculate the accuracy.

As the accuracy evaluator receives different predictions through time, it stores the amount of times it gives predictions to obtain the averages values.

The formal definition for the atomic models is as follows:

prediction\_generator: <X, Y, S, δint, δext, λ, ta>, where:

* X= { in: time\_series\_element }
* Y= { out: time\_series\_element }
* S = {
* last\_price: time\_series\_element
* real\_value: time\_series\_element
* prediction: time\_series\_element
* output\_needed: boolean

}

* δint : any state -> output\_needed=false
* δext:

real\_value = element in X in position with larger date

last\_price = element in X with the day prior to the larger date

prediction = call tensorflow model function with input (X[0..59])

output\_needed= true

• λ: out ={last\_price, real\_value, prediction}

* ta: if (output\_needed == true) {ta=0} else {ta= infinity}
* The input bag and output bag is defined as a set of time\_series\_elements.
* The state is composed by 3 time\_series\_element values, the last price, real value and predicted date and price. Also an output\_needed boolean variable is defined.
* The internal function simply changes the output\_needed variable to false.
* In the external function the interaction between the machine learning algorithm and the DEVS model is used to update the state with the prediction, the DEVS model calls a function that uses a deep learning model previously trained. The function is called with the first 60 elements on the input X.
* On the implementation, this is done through code inside the C language used by Cadmium that calls a function written in Python language written with the Tensorflow and Keras libraries. The function is written in the file ***predictor.py,*** which uses the files ***trained\_model.h***, and ***scaler.save*** previously defined when the deep learning model was trained.
* The output function transmits to the output port, the last\_price, real\_value and prediction.
* The time advance function sets the time advance to 0, when the output\_needed is true to trigger an internal transition and the time advance to infinity when the output\_needed is equal to false to passivate the model.

The accuracy\_evaluator model takes as input a set of 3 elements of time\_series\_elements type: last\_value, real\_value and prediction. As mentioned before, this model returns as output 2 float values, the first one, called average\_error\_estimation is a value indicating the average error in percentage of the estimations made. The second one, accurate\_decision\_percentage tracks the percentage of the correct decisions made by the model. This means, when the stock went higher in price the recommendation made by the model was to sell the stock, when the stock price went lower the decision support system recommended to sell and when the stock stood at the same price, the recommendation was to hold the stock. This recommendations are compared with the real value for that day to calculate when the recommendation with the prediction was correct. The formal description for the model is as follows:

accuracy\_evaluator <X, Y, S, δint, δext, λ>, where:

* X= { in: time\_series\_element}
* Y= { out: time\_series\_element}

• S = {

* last\_value: time\_series\_element
* real\_value: time\_series\_element
* prediction: time\_series\_element
* values\_predicted :integer
* right\_decisions: integer
* accurate\_decision\_percentage: float
* average\_estimation\_error: float
* output\_needed: bool

}

* δint : { any state -> output\_needed=false}
* δext: {

/\* estimates percentage error compared with real\_value, the error goes from 0 to infinity, where 1 corresponds to a 100% error \*/

* estimation\_error=abs(state.prediction.price-state.real\_value.price)/ state.real\_value.price;

/\* accumulates the average estimation error \*/

* average\_estimation\_error=((average\_estimation\_error\*values\_predicted)

+estimation\_error)/(values\_predicted+1);

/\* if the prediction achieves to follow the trend of the real\_value, grows, dimishies or holds then the model will make a good decision

\*/

* if (real\_value=last\_value && estimation = last\_value) {

right\_decisions++;

}

else { if (real\_value>last\_value && estimation > last\_value)

right\_decisions++

else{

if(real\_value < last\_value && estimation < last\_value)

right\_decisions++

}

}

/\* keeps track of how many predictions have been made \*/

* values\_predicted++;

/\* updates the accurate decision percentage\*/

* accurate\_decision\_percentage= right\_decisions/values\_predicted;
* /\* indicates that an external transition has occur and an internal transition must be triggered \*/

output\_needed=true;

}

* λ: return(random\_value, last\_element)
* ta: if (output\_needed == true) { ta=0} else {ta= infinity}
* The input bag and output bag is defined as a set of time\_series\_elements.
* The state is composed by: 3 time\_series\_element values, the last price, real value and predicted date and price; 2 integers: values\_predicted, to store the amount of values predicted by the model and right\_values to store the amount of right decisions made by the model; 2 float values: accurate\_decision\_percentage to store the percentage of right decisions made by the model and average\_estimation\_error to store the average estimation error made by the system Also an output\_needed boolean variable is defined.
* The internal function simply changes the output\_needed variable to false.
* In the external function, the state is updated.
* First, the estimation error is calculated with the following formula:

***estimation\_error=abs(state.prediction.price-state.real\_value.price)/ state.real\_value.price;***

This formula calculates the percentage of the relative difference between the real and the predicted values.

* Second, the average estimation error is updated with the following formula:

***average\_estimation\_error=((average\_estimation\_error\*values\_predicted)+***

***estimation\_error)/(values\_predicted+1);***

This formula updates the average\_estimation\_error with the new estimation error previously calculated.

* In third place, the decision that is made by the model with the prediction is calculated and compared with the accurate decision that should have been made with the real value is compared with the following algorithm:

***if (real\_value=last\_value && estimation = last\_value) {***

***right\_decisions++;***

***}***

***else { if (real\_value>last\_value && estimation > last\_value)***

***right\_decisions++***

***else{***

***if(real\_value < last\_value && estimation < last\_value)***

***right\_decisions++***

***}***

***}***

This algorithm checks if the estimated value follows the same trend as the real value.

If so, the decision made by the model is correct and the right\_decisions variable is incremented.

* In fourth place, the accurate\_decision\_percentage is updated with the following formula:

***accurate\_decision\_percentage= right\_decisions/values\_predicted;***

* Finally, the values\_predicted variable is incremented and the out\_put\_needed variable is set to true to trigger an internal transition on the next simulation step.
* The output function transmits to the output port, the accurate\_decision\_percentage and average\_estimation\_error values.
* The time advance function sets the time advance to 0, when the output\_needed is true to trigger an internal transition and the time advance to infinity when the output\_needed is equal to false to passivate the model.

The formal definition for the complete coupled models is the following:

Stock Price Simulation:

* X = {in: set of time\_series\_element}
* Y = {out: set of time\_series\_element}
* D = {prediction\_generator, accuracy\_evaluator}
* Md = {I\_prediction\_generator, I\_accuracy\_evaluator}
* EIC = {(self.in, prediction\_generator\_in)}
* EOC = {(accuracy\_evaluator.out, self.out)}
* IC = {(predicition\_generator\_out, accuracy\_evaluator\_in)

# EXPERIMENTAL RESULTS

An experimentation study was conducted to test the proposed model. In first place, the Deep Learning model was trained with stock prices from the Google company from the past 5 years, from 2015 to 2019. This data in on the file GOOG.svc located in the machine\_learning folder. The model can be trained executing the following command inside the machine\_learning folder: “python3 train.py”.

To test the Cadmium implementation different experiments were conducted over the modules to test the correct behavior of each one of them in a bottom up strategy. First the atomics models (prediction\_generator and accuracy\_evaluator) were tested, and finally the complete Stock Price Simulation model.

For the prediction\_generator model, the experiments with the following inputs were performed:

1. 61 Google stock values from 2019-10-01 to 2019-12-26, which can be seen in the input file: “prediction\_generator\_input\_test1”. In this case the value to predict is the price for the day 61 in the sequence: 1346.17

This experiment is executed using the script “prediction\_generator\_test1” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “2019-12-24 1348.5, 2019-12-26 1346.17, 2019-12-26 1349.57”.

It can be seen that the model behaves as expected, the past price, real value and prediction are transmitted through the port.

1. 61 Google stock values from 2019-10-02 to 2019-12-27, which can be seen in the input file: “prediction\_generator\_input\_test2”. In this case the value to predict is the price for the day 61 in the sequence: 1362.99

This experiment is executed using the script “prediction\_generator\_test2” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “2019-12-26 1346.17, 2019-12-27 1362.99, 2019-12-27 1351.73”.

It can be seen that the model behaves as expected, the past price, real value and prediction are transmitted through the port.

1. 61 Google stock values from 2019-10-03 to 2019-12-30, which can be seen in the input file: “prediction\_generator\_input\_test3”. In this case the value to predict is the price for the day 61 in the sequence: 1350

This experiment is executed using the script “prediction\_generator\_test3” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “2019-12-27 1362.99, 2019-12-30 1350, 2019-12-30 1353.22”.

It can be seen that the model behaves as expected, the past price, real value and prediction are transmitted through the port.

For the accuracy\_evaluator model, the experiments with the following inputs were performed:

1. 3 time\_series\_element values: 2018-10-25 1, 2018-10-26 1, 2018-10-26 2, which are saved on in the input file: “accuracy\_evaluator\_input\_test1”.

This experiment is executed using the script “accuracy\_evaluator\_test1” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “0, 1”.

It can be seen that the model behaves as expected, the first value, 0, is the percentage of correct decisions made by the support decision system, in this case the value of the stock did not change from the previous day, and the prediction was an increase in the value, therefore the decision was not correct. 0 represents 0% of correct decisions. The second value, 1, is the average error in the estimation. In this case only one value is predicted, the real value was 1 and the prediction was 2, thus, the error was 1, equivalent to 100%.

1. 3 time\_series\_element values: 2018-10-25 2, 2018-10-26 1, 2018-10-26 1, which are saved on in the input file: “accuracy\_evaluator\_input\_test2”.

This experiment is executed using the script “accuracy\_evaluator\_test2” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “1, 0”.

It can be seen that the model behaves as expected, the first value, 1, is the percentage of correct decisions made by the support decision system, in this case the value of the stock decrease from the previous day, and the prediction was that the value will decrease, therefore the decision was correct. 1 represents 100% of correct decisions. The second value, 0, is the average error in the estimation. In this case only one value is predicted, the real value was 1 and the prediction was also 1, thus, the error was 0, equivalent to 0%.

1. 3 time\_series\_element values: 2018-10-25 2, 2018-10-26 2, 2018-10-26 1, which are stored on in the input file: “accuracy\_evaluator\_input\_test3”.

This experiment is executed using the script “accuracy\_evaluator\_test3” in the folder “scripts”

And the result obtained in the message log file in the output port for this model is the following: “0, 0.5”.

It can be seen that the model behaves as expected, the first value, 0, is the percentage of correct decisions made by the support decision system, in this case the value of the stock maintained its value from the previous day, and the prediction was that the value will decrease, therefore the decision was not correct. 0 represents 0% of correct decisions. The second value, 0.5, is the average error in the estimation. In this case only one value is predicted, the real value was 2 and the prediction was also 1, thus, the error was 0.5, equivalent to 50%.

Finally, to test the correct behavior of the complete model a larger experiment was conducted to evaluate the prediction for the Google stock price for the last 7 business days of 2019:

1. 2019-12-19
2. 2019-12-20
3. 2019-12-23
4. 2019-12-24
5. 2019-12-26
6. 2019-12-27
7. 2019-12-30

For each one of these dates, the past 60 prices were given to the model at different times and on each one of them a prediction was obtained and compared with the real value on that day.

A simplified version (showing only the output ports for the models defined) of the results in the message log file from executing this experiment is the following:

***00:00:00:000 [prediction\_generator\_defs::out: {2019-12-18 1356.6, 2019-12-19 1351.82, 2019-12-19 1334.3}] generated by model prediction\_generator1***

***00:00:00:000 [accuracy\_evaluator\_defs::out: {1, 0.0129631}] generated by model accuracy\_evaluator1***

***00:00:01:000 [prediction\_generator\_defs::out: {2019-12-19 1351.82, 2019-12-20 1363.35, 2019-12-20 1338.79}] generated by model prediction\_generator1***

***00:00:01:000 [accuracy\_evaluator\_defs::out: {0.5, 0.0154888}] generated by model accuracy\_evaluator1***

***00:00:02:000 [prediction\_generator\_defs::out: {2019-12-20 1363.35, 2019-12-23 1355.87, 2019-12-23 1342.94}] generated by model prediction\_generator1***

***00:00:02:000 [accuracy\_evaluator\_defs::out: {0.666667, 0.0135053}] generated by model accuracy\_evaluator1***

***00:00:03:000 [prediction\_generator\_defs::out: {2019-12-23 1355.87, 2019-12-24 1348.5, 2019-12-24 1346.59}] generated by model prediction\_generator1***

***00:00:03:000 [accuracy\_evaluator\_defs::out: {0.75, 0.0104832}] generated by model accuracy\_evaluator1***

***00:00:04:000 [prediction\_generator\_defs::out: {2019-12-24 1348.5, 2019-12-26 1346.17, 2019-12-26 1349.57}] generated by model prediction\_generator1***

***00:00:04:000 [accuracy\_evaluator\_defs::out: {0.6, 0.00889123}] generated by model accuracy\_evaluator1***

***00:00:05:000 [prediction\_generator\_defs::out: {2019-12-26 1346.17, 2019-12-27 1362.99, 2019-12-27 1351.73}] generated by model prediction\_generator1***

***00:00:05:000[accuracy\_evaluator\_defs::out: {0.666667, 0.00878564}] generated by model accuracy\_evaluator1***

***00:00:06:000 [prediction\_generator\_defs::out: {2019-12-27 1362.99, 2019-12-30 1350, 2019-12-30 1353.22}] generated by model prediction\_generator1***

***00:00:06:000 [accuracy\_evaluator\_defs::out: {0.714286, 0.007871}] generated by model accuracy\_evaluator1***

As it can be seen in the log from the execution, every millisecond a new prediction is made by the prediction\_generator model and the accuracy\_evaluator model changes its output parameters accordingly.

The averages for the metrics are calculated step by step and the accumulated values are show at the end. The result obtained was an average estimation error of 0.78% and the accurate decision percentage obtained was 71%. This experiment can be executed with the stock\_price\_simulation.sh script, as described on the README file.

# CONCLUSIONS

The model proposed in this work uses simulation model to evaluate how a decision support system will perform using a machine learning prediction. The results obtained show how the predictor used gives an accurate estimation for future prices and also the decision support system can give good advice to investors. Also, the framework proposed may allow to test the accuracy of different deep learning models and different decision support systems. As a future work we propose to continue analyzing different deep learning models to find the most accurate one to predict future prices and give more accurate recommendations to investors.

# references

Antoine-Santoni T., Poggi B., Vittori E., Van Hieux H., Delhom M., Aiello A. 2019. “Smart Entity” – How to Build DEVS Models from Large Amount of Data and Small Amount of Knowledge?. In: Song H., Jiang D. (eds) Simulation Tools and Techniques. SIMUtools 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 295. Springer, Cham

Saadawi H., G. Wainer and G. Pliego, 2016. "DEVS execution acceleration with machine learning," Symposium on Theory of Modeling and Simulation (TMS-DEVS), Pasadena, CA, 2016, pp. 1-6.

Mahmoud Elbattah and Owen Molloy. 2018. ML-Aided Simulation: A Conceptual Framework for Integrating Simulation Models with Machine Learning. In Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '18). ACM, New York, NY, USA, 33-36.