Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

ENABLING INTELLIGENT PROCESSES IN SIMULATION UTILIZING THE TENSORFLOW DEEP LEARNING RESOURCES

Rodrigo De la Fuente
Ignacio Erazo

Department of Industrial Engineering
University of Concepción
Edmundo Larenas 219
Concepción, CHILE

Raymond L. Smith III

Department of Engineering
East Carolina University
232 Slay Building
Greenville, NC 27858, USA

ABSTRACT

Availability of large data sets and increased computing performance have contributed to many improvements in productivity and decision-making. Simulation can exploit these by incorporating data mining capabilities, such as machine learning, in the modeling and analysis process. This paper demonstrates the integration of discrete event simulation with a deep learning resource, known as TensorFlow, to enable intelligent decision making in the form of smart processes. A bank credit approval process is modeled using these smart processes to evaluate customer credit worthiness based on 20 reported features. Comparison of three models is made where credit worthiness is (1) known, (2) randomly assigned, or (3) evaluated based on customer features. Additionally, the experiment compares results under conditions where the process is perturbed by an unexpected surge in customer arrivals. The presented models and results demonstrate the feasibility of enabling smart processes in discrete event simulation software and the improved decision-making fidelity.

1 INTRODUCTION

At the core of most competitive and successful organizations is a data driven decision-making process. Currently, the increased computational power, improvements in storage performance and reduction in data collection costs have resulted in the emergence of the big data era. Effectively using the available data to develop valuable insights is the challenge organizations presently face. Discrete event simulation is a powerful tool organizations can use to study a system or process over time with the ability to record measurements and prepare for the uncertain future.

One of the more notable difficulties in modeling is being able to reproduce the essential system behavior while obtaining accurate predictions while utilizing a model that is simple enough to run. In real world systems and processes, the flow of materials, information or people enter the system at some point in time and behave according to their unique characteristics. Traditionally entities in simulations do not carry much information mainly because it is difficult to implement complex decision processes inside of simulation software. In fact, discrete event simulation software usually creates entities and their respective properties based on probability distributions for every variable, and server resources make decisions using structured rules instead of taking into account these special features.

The main objective of this work is to demonstrate how it is possible to build a smarter simulation, by integrating machine learning algorithms within the simulation process. The result of this hybrid methodology is a model that more closely resembles reality. The demonstration is completed using open-source software. Thus, inside the simulation models presented in this work, the branching step will be performed by a machine learning algorithm which provides a better classification via the information attached to each customer. The paper is organized in the following manner. First, a brief literature review is presented to

contextualize how this work fits into the state-of-the-art. Second, the methodology explains in detail the steps followed to build the machine learning model, the simulation model and the experiment. Third, the results are presented, and the main findings are discussed by the authors to assess how beneficial it might be for a researcher to invest time and effort implementing a similar hybrid methodology. Last, a conclusion and future work are given.

2 LITERATURE

This section presents a brief literature review to contextualize how this work fits into the state-of-the-art. Historically, advancements in stochastic simulation software capability, feature and function have trailed the advancements in computer technology. In earlier days, simulation practitioners had to carefully consider the computational limitations regarding their modeling and analysis projects since processing was performed on a single processor and data storage was minuscule. Additionally, this often-meant data captured from the simulation execution was limited to summary detail. At present, advancements in computer technology, particularly related to increased processing capability and scalable data storage, have vastly outpaced the improvements for stochastic simulation.

Recognizing this imbalance Nelson (2016) describes in detail the tactical problems facing the simulation community in the next decade. Notably, the author makes three central observations. First, the availability of inexpensive data storage provides an opportunity to exploit the simulation-generated data unlike in the past. Second, parallel simulation offers significant benefits for simulations presently requiring multiple replications or multiple scenarios. Third, simulation users are increasingly interested in utilizing stochastic simulation for risk analysis, prediction and control. Elements of Nelsons observations are directly related to the work presented here, as it depends upon recent computer technology advancements to implement the intelligent processes within a stochastic simulation. A vital component, not discussed among Nelson's observations, is the machine learning functionality, specifically deep learning, used to perform the decision making.

Machine learning uses algorithms to parse data, learn from the data, and make informed decisions based on what has been learned. Deep learning, a sub-field of machine learning, structures algorithms in layers to create an artificial neural network capable of learning and making intelligent decisions. In the subsections that follow, a brief background for artificial neural networks is presented and recent applications considered relevant to stochastic simulation are cited.

2.1 Artificial Neural Network (ANN)

Artificial neural networks (ANN) were motivated by the biological neural networks used to process information in the human brain. Inspired by the biological neuron, McCulloch and Pitts (1943) proposed a computational model for artificial networks using mathematics and algorithms, where decision making is based on a concept known as threshold logic. Using this approach, a neural network is designed to work by classifying information in a similar way a human brain does. Neural networks can be taught to recognize, for example, images, and classify them according to elements they contain. In this way, artificial neural networks are flexible function approximators well suited to replicate behavior of systems otherwise too difficult to model mathematically (Ripley 2007). In its simplest form, artificial neural networks are comprised of layers of simple computational entities, resembling the biological neutrons in the brain, which are linked to other neurons by weighted connections. Each neuron has an activation level which depends on the activation of neighboring connected neurons. In determining an activation, every neuron must summarize its inputs with respect to the corresponding connection weightings. This calculated sum is then transformed by an activation function which generates the corresponding output behavior.

Software used to develop artificial neural networks has evolved from individually written code to cloud based solutions enabled for big data. In 2017 Google released an open source state-of-the-art machine learning platform known as TensorFlow (Abadi et al. 2016). TensorFlow can run on multiple processors

and provides the ability to perform general purpose computing on graphical processing units. This makes TensorFlow one of the most powerful machine learning platforms available. The name TensorFlow is derived from the operations that neural networks perform on multidimensional data arrays.

2.2 Simulation Applications

Over the past two decades researchers have sought to incorporate machine learning into simulation to improve speed of execution, increase estimation accuracy, and reliable behavior. It is possible to categorize many of these simulation applications as being related to: 1) metamodeling and analysis, 2) parameter prediction, and 3) intelligent decision making. Applications of each are discussed below.

First, metamodeling is an approach commonly used to develop a representative model of a model which may be very computationally expensive to execute. Early implementations utilized statistical approaches such as regression and surface response models to develop their representative models. Although useful, the range of applications these representative models can provide is limited. Kilmer and Smith (1993) demonstrates for a small number of input parameters and output measures the ANN approach can outperform the first and second order linear regression models typically used in simulation response surface methods. Additionally, Kilmer et al. (1999) demonstrates that using the ANN approach can be quite competitive in accuracy when compared to the simulation and operate in nearly real-time. Fonseca et al. (2003) presents the importance of simulation metamodeling using ANN and develops a series of guidelines to assist simulation practitioners in the design of intelligent systems. Alternative approaches challenging the robustness of ANN have been made over the years. A comparative study by Can and Heavey (2012) found that while genetics algorithms may achieve greater accuracy during training validation stage they require more computation in the metamodel development stage than ANN. Improved accuracy provided by ANN has led to increased use of metamodels for perform sensitivity analysis. Finally, an updated contemporary comparison of machine learning algorithms applied to simulation metamodeling can be found in De la Fuente and Smith (2017).

Second, parameter prediction and estimation for use within the simulation execution is a recent development. In this case, a trained ANN provides parameter predictions when demanded by the simulation model. Using a hybrid approach, Elbattah and Molloy (2016) demonstrates the coupling of a simulation model with the Microsoft Azure machine learning environment to make predictions for hip replacement surgery on the inpatient length of stay and discharge destination of simulation generated patients. This information is subsequently used to develop demand predictions on healthcare resources, which impacts future patient scheduling. Additionally, Chang and Chang (2018) demonstrates the use of ANN to make predictions about the treatment duration of individual patients in a dental clinic setting in conjunction with a discrete event simulation for evaluating appointment scheduling performance. The improved treatment duration predictions have resulted in improved appointment scheduling.

Last, intelligent decision-making within stochastic simulation model execution represents the newest evolution; it also has the fewest number of published examples at this writing. Silva et al. (2016) demonstrates the integrated use of an ANN in a discrete event simulation model, constructed using open source software, to mimic the human behavior and decision-making of a traffic control officer directing vehicles at an intersection; however, the analysis specific to performance is incomplete. Bergmann et al. (2017) utilizes a supervised machine learning approach to develop the approximation of dispatching rules used for decision-making within a discrete event simulation model. The author mainly focuses on comparing the most common supervised machine learning algorithms, such as K-nearest neighbors, Nave Bayes classifier, support vector machines, decision trees, and artificial neural networks. These limited published works leave many opportunities and concerns relevant to intelligent decision making embedded in simulation unaddressed. The presented work seeks to contribute to the general body of knowledge and improve the general understanding needed for more sophisticated implementations.

3 METHODOLOGY

This section describes the methodology used to construct the simulation model and the machine learning function. As previously stated, the aim of this work is to enable intelligent decision-making capability within the discrete event simulation model using a machine learning algorithm. It is expected that the algorithm will be able to address unanticipated changes occurring in the data generation process. In the simulation model, entities possess a unique set of attributes, or features, used to determine what action should be applied to them when requesting a server resource. As feature detail increases -contributing to both system realism and complexity- the use of an advanced decision-making approach, such as a machine learning algorithm, becomes attractive to enhance and improve results. However, limited understanding exists regarding differences between traditional and algorithm-based decision-making with respect to system performance and outcomes.

To investigate these differences, an experiment is formulated to compare the outcomes from three variations of the base simulation model, which include: 1) a reference model where decisions are predetermined such that outcomes are known with certainty; 2) a classic model where decisions are based on probabilistic hard-branching that is characteristic of most discrete event simulations; and, 3) a smart model where decisions are determined by the machine learning algorithm based on the features presented by the entities. For comparison purposes two time-persistent statistics are captured: 1) the average number of customers in the system, and 2) the average number of customers in queue after a pre-specified running time t. Additionally, the simulation model must be able to integrate the smart decision-making object to allow the evaluation of features assigned to each entity, thus providing an appropriate benchmarking between the hybrid approach using smart decision-making and those approaches that do not.

The remainder of this section is organized into four subsections describing: 1) the system modeled and the simulation development process, 2) the data acquisition, its analysis and integration to the simulation model, 3) the enablement of the machine learning algorithm, and 4) the general procedure used to condition the algorithm and execute the experiment. Results from the experiment will be presented in the next section.

3.1 Simulation Model

The simulation model consists of modeling the credit scoring process in a bank situation as presented in Figure 1. As shown in the figure, the customers arrive at the bank according to a non-homogeneous Poisson process with arrival rates of 8, 14, and 20 people per hour. These arrival rates are evenly distributed across three 180-minute blocks of time over the banking day. The system starts the banking day empty and idle and ends with the closing time event when the last customer leaves the system. The simulation models were built using SimPy 3.0.8, an open-source hybrid-simulation suite developed entirely on Python (Matloff 2008). Additionally, Figure 1 shows a sketch of the classes and their encapsulation. The *Model* object governs the entire simulation process, which contains an *Entrance* object used to generate *Customers* according to a non-homogeneous Poisson process and assigns credit features to each of them. Within the *Model* object a decision process is implemented that describes the route a customer will take and the servers they will engage (SimPy resources) as credit worthiness is determined. If the customer receives a credit, they depart the system and the appropriate statistics are collected. However, if the customer is denied credit, the customer will be routed to a loan specialist (SimPy resource) where further inquiry is made about the customers financial situation and a request form is completed. Finally, the customer then departs the system and the appropriate statistics are collected.

3.2 Data

With the simulation framework complete, the next step is to find adequate data to build a more realistic version of the simple model. Since the model is about obtaining a loan from a bank, a data set containing historical information regarding customer credit ratings was needed. In the machine learning literature,

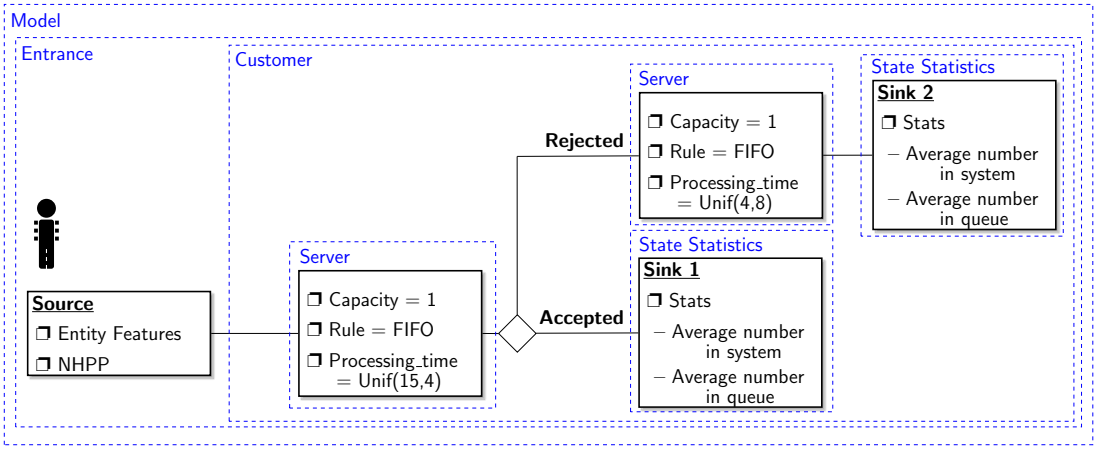

Figure 1: Simulation model structure.

a classic dataset containing this information is the "German dataset", available in the *Machine Learning Repository* at the University of California-Irvine (UCI 2017).

The dataset consists of 1000 observations, where each row contains a vector with 21 elements, with the first 20 elements of an observation being information provided by the customer applying for the loan, and the last element being the final decision made by the clerk. Additionally, the proportion of final decision outcomes is observed to be unbalanced given 70% of customers received credit and 30% did not. A detailed view of the dataset is given on Figures 2 and 3, which show the correlation between pairs of variables and the scatter point distributions between some selected variables respectively. From Figure 2 it is possible to see that no strong correlation exists either between variables or between variables and the response. The only exception is the relationship between the requested loan amount and the duration which is logical since the greater extension of credit requires more time for repayment. Likewise, Figure 3 shows the relationships between some selected pairs of features. The coloring reveals no clear structure regarding how the features separate the categories. A detailed step-by-step analysis can be found in the website of the *Applied Data Mining and Statistical Learning* course at Pennsylvania State University (PSU 2017) where several algorithms are tested on the same dataset ranging from 50% to 75% accuracy; although, the author does not have to control that the unbalancedness of the data is preserved to have representative entities in the simulation.

The idea is to use this data set to describe the loan seeking customer characteristics and their final decision outcome. The data is partitioned using repeated stratified k-folds (where $k = 10$ and the number of repetitions is 3, totaling 30 training-testing pairs). For each of these pairs the machine learning algorithm is trained (outside the simulation) using the training set, and then the testing set and the trained object are passed to the simulation to assign "unseen" features to the entities and perform smart branching respectively. Maintaining the proper balancing of the decision outcomes used to characterize each of the entities in the simulation is especially important for the testing set. This will allow the three model variations implemented in the simulation to be tested using two different approaches controlling the feature assignment made during the creation of each entity. The approaches include where: 1) the features assignment is made according to the rows of the testing set as specified by the cross-validation process - this maintains proportionality and guarantees randomization; and, 2) the feature assignment is made by sorting the testing set from creditable to non-creditable customers and then assigning rows to newly created entities from top-to-bottom, which induces a sudden change in the arrival pattern of each type of client.

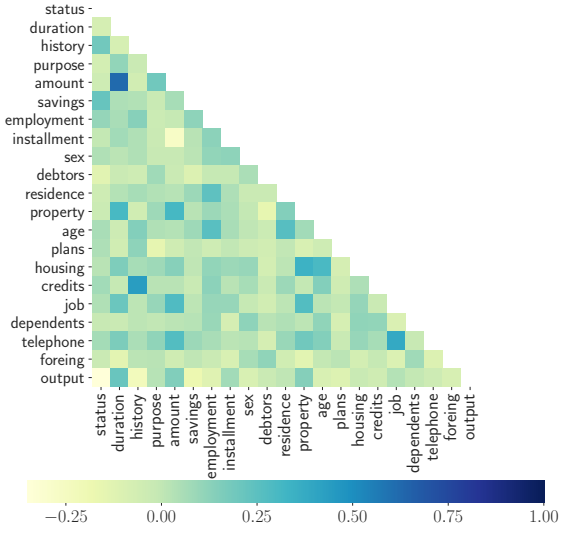

Figure 2: Correlation german dataset.

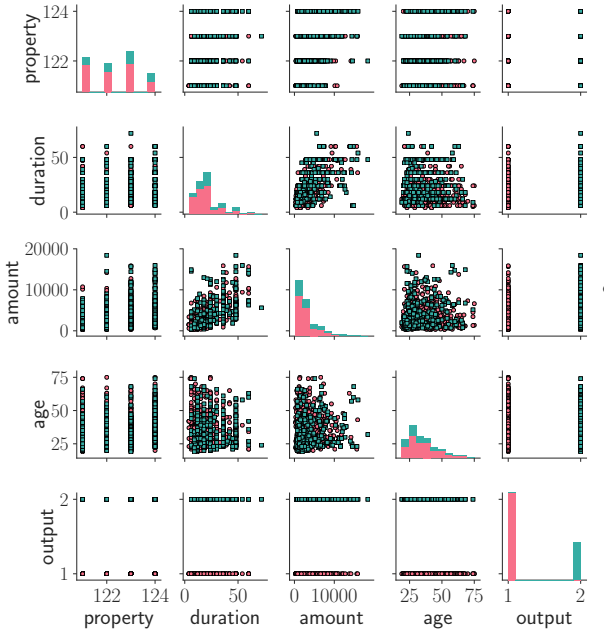

Figure 3: Pair plots selected variables.

3.3 Machine Learning Algorithm

As stated at the beginning of this section, the aim of this work is to evaluate the performance of a smart simulation, where smart means that some part of the decision process -such as the branch decision-making- is performed by a machine learning algorithm. Even though the dataset is small and a neural network with few layers should suffice, to develop a more extensible modular model the machine learning suite TensorFlow (Abadi et al. 2016), generally used for deep learning, is utilized with the aim of showing that more complex datasets can be tackled the same way. Additionally, as suggested in the website of the *Machine Learning Repository* (UCI 2017) a cost matrix is used to address the unbalancedness of the data. In our scenario, it is more undesirable to classify a customer as a good credit risk when they are in fact a bad credit risk rather than the other way around. Therefore, a custom cost function with penalties 5 and 1 are assigned, respectively, for the two situations described. This is easily implemented in TensorFlow. The neural network implementation in this work is presented in Figure 4. It consists of an input layer, three hidden layers with 100 neurons each, and an output layer. A rectified linear unit (ReLu) activation function is used for the first three layers, and for the last layer "softmax" is used to represent the probability distribution of possible outcomes. Then, the weighted cost function is optimized during 5000 epochs with initial learning rate of 1e-4 and batches of 100 elements inside the *train* module using Adam optimizer which performed better than gradient descend. Figure 5 illustrates, as an example, the computational graph steps taken by gradient descend inside the *train* object where the gradients are computed for each variable. Finally, techniques such as node dropout with probablity set to 0.5 for all hidden layers and L-1 regularization are implemented to control overfitting.

3.4 General Procedure

Algorithm 1 provides a summary description in pseudocode form of the computational steps followed. First, the data is read and stored on a Pandas DataFrame. Then, 30-fold cross-validation steps are taken using a random stratified sampling approach to ensure the even representation of classes. Inside the for-loop general preprocessing steps are completed using *sklearn* (Pedregosa et al. 2011), the TensorFlow graph

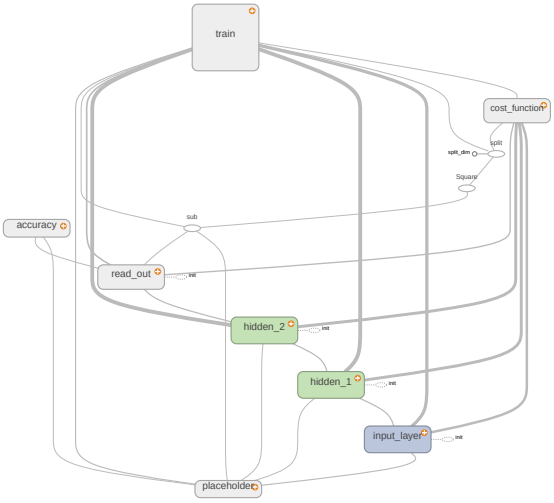

Figure 4: Neural network in TensorBoard.

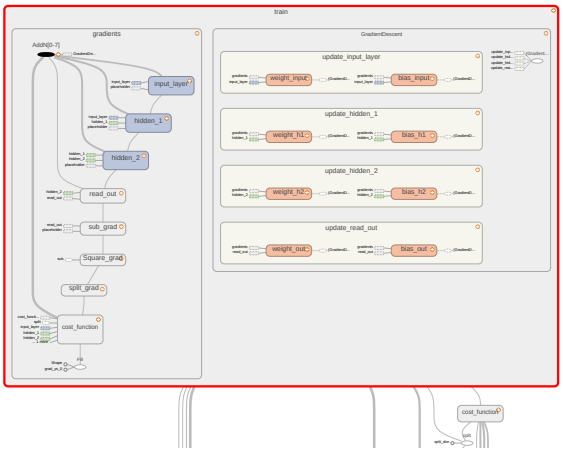

Figure 5: Gradient descent display.

presented on Figure 4 is fitted -using the training set- and, then, the fitted object, the testing set, and the preprocessing *sklearn* object are passed as arguments to the simulation as described in Section 3.2.

Using the same random seed streams, the simulation runs the experiment for the reference model, classic model and smart model configurations. All relevant statistics are captured and saved for further analysis. Information regarding out-of-sample accuracy metrics are collected from the neural network and saved for further analysis. Finally, the simulation results and the out-of-sample cross-validation results are analyzed using StatModels open-source software available in Python (Seabold and Perktold 2010). The findings are discussed in the next section on results.

Algorithm 1: Computational steps.

```
raw_data ← read.txt(German.data.txt);
df_data ← pd.Dataframe(raw_data, Column_names);
stratified_folds ← Repeated_Stratified_CV(df_data);
simulation results ← [ ]
performance results ← [ ]
for train_indices, test_indices in stratified_folds do
    train_set ← df_data[train_indices]
    test_set ← df_data[test_indices]
    mapper ← Mapper(mapper_rules)
    transf_train_set ← mapper.fit_transform(train_set)
    tf ← tf_NeuralNet(**args).fit(transf_train_set)
    result_replicate ← Simulation_Model(mapper, test_set, tf, **args)
    simulation_results.append(result_replicate)
    transf_test_set ← mapper.fit_transform(test_set)
    prediction ← tf.predict(transf_test_set.X)
    accuracy_measures ← accuracy(transf_test_set.Y, prediction)
    performance_results.append(accuracy_measures)
result_summary ← process_sim_results(simulation_results)
performance_summary ← process_perf_results(performance_results)
```

4 RESULTS

The results presented in this section are organized into two subsections: 1) the neural network metrics, and 2) the simulation model. All computations were performed using a Microsoft Windows Surface Book 2 personal computer running the Windows 10 operating system, and configured with 16 gigabytes of RAM, an i7-8650U multi-core 8^{th} generation microprocessor, and a NVIDIA GeForce GTX 1060 graphic processing unit (GPU) with 6 gigabytes of GDDR5 RAM. The neural network training was parallelized to the GPU which was CUDA enabled (Nickolls et al. 2008), and random batches were used to train the gradient descent algorithm for 3000 epochs. The three different simulation scenarios were run sequentially.

4.1 Neural Network Metrics

The main metrics to test in classification problems, such as the one posed by the German dataset, include: 1) precision, 2) recall, and 3) f-1 score. Precision responds to the question, "how many of the selected items are relevants?", whereas, recall responds to "how many of the relevant items are selected?". Given these metric, the f-1 score which represents their harmonic mean can be derived. These metrics fall in the range $[0, 1]$. Table 1 shows the average values over the 30-folds cross-validation. Results illustrate that the neural network achieved values within the range presented on the website of the *Applied Data Mining and Statistical Learning* course (PSU 2017); thus, the customized cost function did not help for controlling the unbalancedness. Additionally, there is a small variation around the mean of each metric which shows that the predicted performance of the results was similar across folds.

Table 1: Neural network out-of-sample metrics.

Criteria	Mean	Std
Precision	0.691	0.048
Recall	0.671	0.048
f-score	0.677	0.049

Another important tool used to test and summarize the accuracy of a classification algorithm is the confusion matrix. Table 2 shows the aggregated results for the 30-folds cross-validation (3000 clients). The results indicate that there is more potential for an error in classifying a bad credit risk as a good credit risk than vice versa, which adds additional information from what was shown in Table 1 and explains what is affecting the metrics. Nonetheless, these results will be utilized to assess the pros and cons of employing a neural network as a smart dispatcher inside of a discrete event simulation.

Table 2: Aggregated out-of-sample confusion matrix.

Criteria	Good credit	Bad credit
Good credit	1783	317
Bad credit	456	444

4.2 Simulation Model

The primary metrics used to evaluate the simulation model include the: average number in system and average number in queue for the time period beginning at 300 minutes. Two general assumptions are made for the arrival process. In the first experiment customers arrive according to the random positional assignment found in the testing set made by the cross-validation object, where the arrival times are assigned sequentially from top to bottom for the dataset. In the second experiment the aim is to evaluate the performance of the smart decision process. This is accomplished by sorting the test dataset by the credit status column. As a

result, creditworthy customers are placed ahead of non-creditworthy customers; however, the arrival times assigned in the first experiment are maintained.

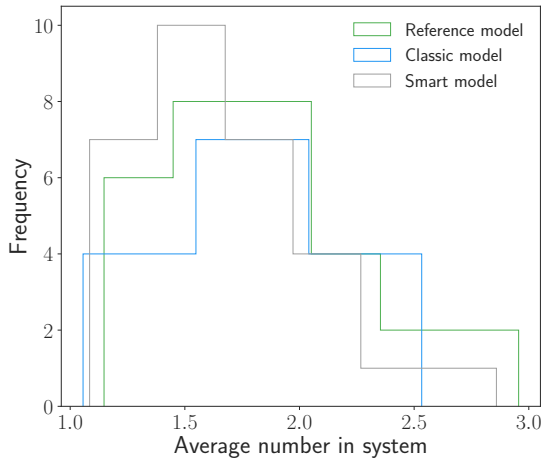

Figure 6: Regular arrivals.

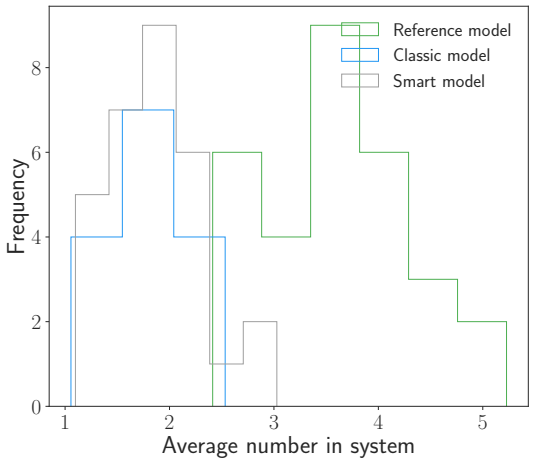

Figure 7: Sorted arrivals.

Figures 6 and 7 show the distribution of the mean number in systems for the reference, the classic, and the smart simulation models on the two arrival arrangements. Figure 6 illustrates that independent of the decision process implemented inside of the simulation, the distribution of the average values over 30 replicates are determined to be statistically indistinguishable as shown in Table 3. However, when the model is perturbed with an unforeseen change, the behavioral pattern changes abruptly, as shown in Figure 7. As expected, the classic simulation case, which is unaware of the customers' features, fails to identify the change in the arrival pattern as seen in Figure 7. Even though results for the smart model are oriented to the right of the classic model, and Table 3 indicates the difference of means between both models to be significant, the smart model also fails to recognize the sudden change in the arrival pattern. This can be attributed to the fitting quality of the machine learning algorithm, as previously discussed in Section 1. Moreover, as presented in Table 3 the statistical difference between the reference model and the others is significant.

Table 3: Difference of means confidence intervals (95% confidence).

	Avg. number in queue, $t \geq 300$		Avg. number in system	
Comparison	Base	Modified	Base	Modified
Classic-Reference	(-0.1303, 0.02139)	(6.0409, 7.0434)	(-0.0891, 0.0496)	(1.6371, 1.9483)
Classic-Smart	(-0.1265, 0.0337)	(0.5906, 2.1094)	(-0.0355, 0.0869)	(0.1660, 0.6689)
Reference-Smart	(-0.0346, 0.05069)	(-5.8784, -4.5058)	(-0.0171, 0.1079)	(-1.5928, -1.1577)

The results for the number in queue at the specialist for $t \geq 300$ are shown in Figures 8 and 9. For the regular arrivals, the behaviors illustrated (in the left-hand figure) for the three simulation model results are observed to be nearly the same - this observation is also substantiated by the results given in Table 3. Again, the random assignment generated by the hard decision rule implemented in the classic model is compensated by the random sorting produced by the cross-validation object. Nonetheless, when the model is subject to a sudden change that affects the specialist server (hence, more people arriving at a specific time) the difference among the models becomes evident. Although the distributional shape of the smart model results shift more toward the reference model than its classic counterpart, again it is possible to

observe the effect that poor fit in the machine learning algorithm has on the smart simulation model by producing mean number in queue values no greater than 3, as illustrated in Figure 9. These results greatly differ when compared with the reference model. Finally, a statistical difference of means test was carried out to corroborate whether the difference was statistical significant or not. Results indicate that none of the means among models are equal as shown in Table 3.

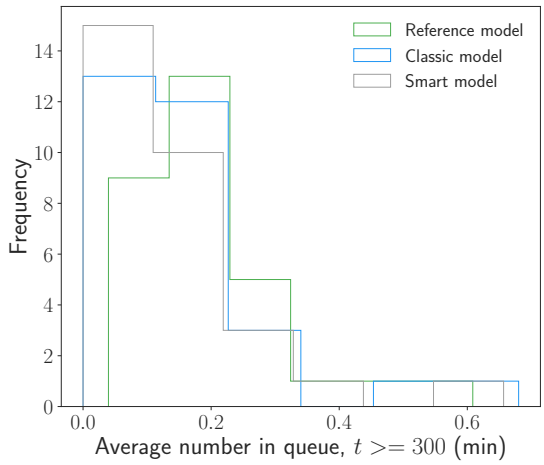

Figure 8: Regular arrivals.

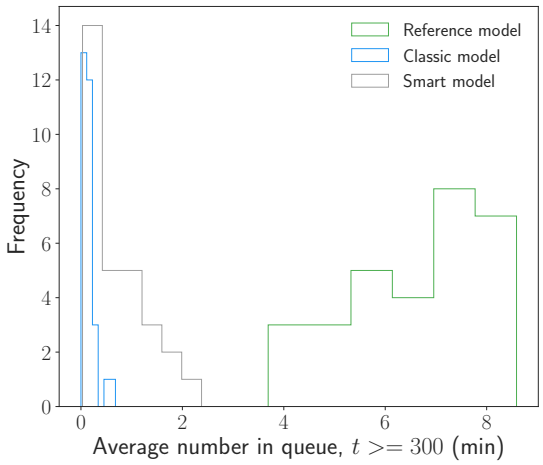

Figure 9: Sorted arrivals.

5 DISCUSSION

The most relevant findings from this work are summarized into three central points of discussion. First, the work presented demonstrates it is feasible to use open-source software to compose a smart simulation by integrating the state-of-the-art machine learning suite, TensorFlow, with a discrete event simulation package. TensorFlow allows this to be easily implemented in Python, but it is not a trivial endeavor and integration with commercial simulation software maybe difficult. By comparison, a conditional tree can be easier to implement in all simulation software, since it consists of explicit if-then-else rules. In this case, the user will make a trade-off in accuracy to obtain flexibility. If the dataset being referenced is large, such as big data, deep learning tools implemented in TensorFlow will facilitate not only a better learning behavior (Sun et al. 2017) but also a faster fitting process due to the potential use of the GPU instead of the CPU unit. Additionally, the use of TensorBoard to visualize the construction and training process of the machine learning algorithm provides a powerful tool for debugging purpouses. With respect to the simulation, many times you do not have to implement huge simulation models, but you want to have flexibility to write a small model, which is what SimPy provides (Nelson 2013), and this work has shown that it can be easily integrated with other Python libraries.

Second, attention must be given to the simulation results using regular arrivals and sorted arrivals. The first case did not show any difference in the metrics evaluated, but this could be attributed to the small size of the model. However, as demonstrated there is no guarantee the smart model will behave exactly like the reference model, due to additional random noise induced by the machine learning algorithm. Thus, a better question to evaluate is how much data would I need to train a neural network with the aim of having a robust fit that mimics the reference model behavior when a decision had to be performed unexpectedly. Since it is not expected that the machine learning algorithm produces a perfect decision boundary for classifying clients, the envisioned null hypothesis should test that the difference of means between the reference and smart models were evaluated to be larger than a certain threshold ($H_0 = \mu_{reference} - \mu_{smart} \geq threshold$).

This last point could be addressed through the generation of synthetic data and repeating the process implemented in this work with datasets of increasing size.

Lastly, it is important to mention that the reference model observable credit scoring classifications were required to test the smart model again. The authors acknowledge that it may be difficult to access this information, however, under such circumstances the features could be obtained from the joint density, generated perhaps using the NORTA method (Cario and Nelson 1997). Due to the curse of dimensionality, these results can be greatly affected by the fitting accuracy of the generation process; thus, a more advanced procedure should be considered (Ghosh and Henderson 2003).

6 CONCLUSION

This work demonstrates it is worth investing effort into building smarter simulation models given the process is now easier than ever before due to the proliferation of powerful open-source software resources. However, a researcher must proceed with caution because the addition of smart machine learning objects may introduce additional error into the simulation process. This may cause the results to be bias, as shown in this work. The authors believe the use of more data, or big data, to fit the machine learning objects would lead to a reduction in the bias; although, the assertion has not been explored here. The work also demonstrates that the implementation of smart decision objects perform equivalent to the classic hard-branching approach when the simulated systems does not experience unforeseen changes. Nevertheless, due to the scarcity of data and the complex hyper-space spanned by it, the smart model could not achieve a comparable performance to the reference model, being unable to adjust to abrupt changes.

Future work includes the implementation of this methodology in an expanded setting capable of testing the results presented here using a larger dataset. Additional work must be performed testing the effect machine learning fitting metrics have on simulation metrics, especially concerning the noise they may contribute to the results. Finally, enhancements must be completed to include the multivariate joint densities as the data generation process coupled with a machine learning algorithm.

REFERENCES

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al. 2016. "TensorFlow: A System for Large-Scale Machine Learning.". In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, 265–283. USENIX Association.

Bergmann, S., N. Feldkamp, and S. Strassburger. 2017. "Emulation of Control Strategies Through Machine Learning in Manufacturing Simulations". *Journal of Simulation* 11(1):38–50.

Can, B., and C. Heavey. 2012. "A Comparison of Genetic Programming and Artificial Neural Networks in Metamodeling of Discrete-Event Simulation Models". *Computers & Operations Research* 39(2):424–436.

Cario, M. C., and B. L. Nelson. 1997. "Modeling and Generating Random Vectors with Arbitrary Marginal Distributions and Correlation Matrix". Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois.

Chang, W.-J., and Y.-H. Chang. 2018. "Design of a Patient-Centered Appointment Scheduling with Artificial Neural Network and Discrete Event Simulation". *Journal of Service Science and Management* 11(01):71.

De la Fuente, R., and R. Smith. 2017. "Metamodeling a System Dynamics Model: A Contemporary Comparison of Methods". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. Chan et al., 1926–1937. Piscataway, New Jersey: IEEE, Inc.

Elbattah, M., and O. Molloy. 2016. "Coupling Simulation with Machine Learning: A Hybrid Approach for Elderly Discharge Planning". In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 47–56. ACM.

Fonseca, D. J., D. O. Navaresse, and G. P. Moynihan. 2003. "Simulation Metamodeling Through Artificial Neural Networks". *Engineering Applications of Artificial Intelligence* 16(3):177–183.

Ghosh, S., and S. G. Henderson. 2003. "Behavior of the NORTA Method for Correlated Random Vector Generation as the Dimension Increases". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 13(3):276–294.

Kilmer, R. A., and A. E. Smith. 1993. "Using Artificial Neural Networks to Approximate a Discrete Event Stochastic Simulation Model". *Intelligent engineering systems through artificial neural networks* 3(1):631–636.

Kilmer, R. A., A. E. Smith, and L. J. Shuman. 1999. "Computing Confidence Intervals for Stochastic Simulation Using Neural Network Metamodels". *Computers & Industrial Engineering* 36(2):391–407.

Matloff, N. 2008. "Introduction to Discrete-Event Simulation and the Simpy Language". *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August* 2(2009):1–33.

McCulloch, W. S., and W. Pitts. 1943. "A Logical Calculus of the Ideas Immanent in Nervous Activity". *The bulletin of mathematical biophysics* 5(4):115–133.

Nelson, B. 2013. *Foundations and Methods of Stochastic Simulation: a First Course*. New York, NY, United States: Springer Science & Business Media.

Nelson, B. L. 2016. "Some Tactical Problems in Digital Simulation for the Next 10 Years". *Journal of Simulation* 10(1):2–11.

Nickolls, J., I. Buck, M. Garland, and K. Skadron. 2008, March. "Scalable Parallel Programming with CUDA". *Queue* 6(2):40–53.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion et al. 2011. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12:2825–2830.

PSU 2017. "Applied Data Mining and Statistical Learning". https://onlinecourses.science.psu.edu/stat857/intro. Accessed April 4th, 2018.

Ripley, B. D. 2007. *Pattern Recognition and Neural Networks*. Cambridge, United Kingdom: Cambridge University Press.

Seabold, S., and J. Perktold. 2010. "Statsmodels: Econometric and statistical modeling with python". In *Proceedings of the 9th Python in Science Conference*, 57–61: Scipy.

Silva, M., J. Rangel, D. Silva, T. Peixoto, and I. Matias. 2016. "Decision with Artificial Neural Networks in Discrete Event Simulation Models on a Traffic System". *Pesquisa Operacional* 36(1):133–149.

Sun, C., A. Shrivastava, S. Singh, and A. Gupta. 2017. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In *2017 IEEE International Conference on Computer Vision (ICCV)*, 843–852. Piscataway, New Jersey: IEEE, Inc.

UCI 2017. "Machine Learning Repository". http://archive.ics.uci.edu/ml. Accessed March 26th, 2018.

AUTHOR BIOGRAPHIES

RODRIGO DE LA FUENTE is an Assistant Professor in the Department of Industrial Engineering at the University of Concepción (Chile). He holds a Ph.D. in Industrial and Systems Engineering from North Carolina State University. His research interests include discrete-event simulation and machine learning. His e-mail address is rodelafuente@udec.cl.

RAYMOND SMITH III is an assistant professor in the Department of Engineering at East Carolina University located in Greenville, North Carolina. He holds a Ph.D. in Industrial and Systems Engineering from North Carolina State University. His research interests include health systems engineering, simulation modeling and project management. His e-mail address is smithraym17@ecu.edu.

IGNACIO ERAZO is a student of the Master's program in Industrial Engineering at the University de Concepción (Chile). He holds a Bachelor's degree in Industrial Engineering from the University of Concepción (Chile). His research interests are data analysis, simulation and optimization. His e-mail address is ierazo@udec.cl.