Reducing response time with data farming and machine learning

Falk Stefan Pappert[1*], Oliver Rose[1]

[1]Fakultät für Informatik, Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, 85579 Neubiberg, Germany * *falk.pappert@unibw.de*

Abstract. In industry, there are numerous applications for simulation. However, simulation in our area usually takes some time even if a preexisting model just needs to be parameterized; there is still the run time, which will usually take at least a few minutes if not hours. In our current case, a planner wanted to know for a given product mix situation and for an equipment group with specific characteristics how much he can utilize the equipment without violating flow factor targets. A question, which arises several times during a typical workday as new orders are coming in and the situation on the shop floor is continuously changing. Since the user is usually asking the same question just with different parameters we are able to solve the waiting time problem while still giving good decision support. Instead of simulating every scenario at the time the user actually needs these answers, we use data farming to generate a large set of data points that are then used to train a neural network. This neural network then substitutes for the simulation and responds to the user immediately.

Introduction

A crucial task in modern industry is capacity planning. Robinson et al. [1] point out why accurate capacity planning is so important, yet so difficult to achieve in the highly sophisticated semiconductor industry. A planner faces numerous questions every day from short-term operative questions to long-term strategic ones. A necessary starting point to make any reasonable decsions is to know the available equipment capacity and how its' utilization influences the material flow.

As cycle times vary from product to product flow factors (cf. Equation 1) are good indictors to evaluate a production system.

$$flow\ factor = \frac{actual\ cycle\ time}{raw\ process\ time} \quad (1)$$

The trade off between utilization and flow factor can be visualized as operating curves (cf. [2]), which relate a system's flow factor against its utilization. Operating curves are an important tool in managing semiconductor fabs (cf. [3]).

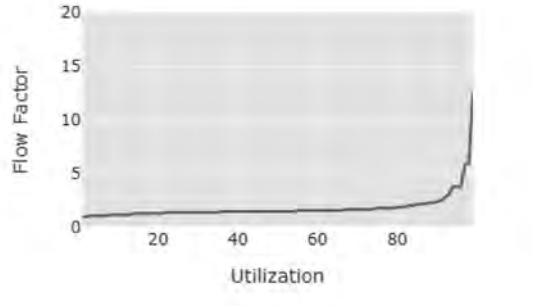

Figure 1: Operating Curve of a basic single equipment without any special features

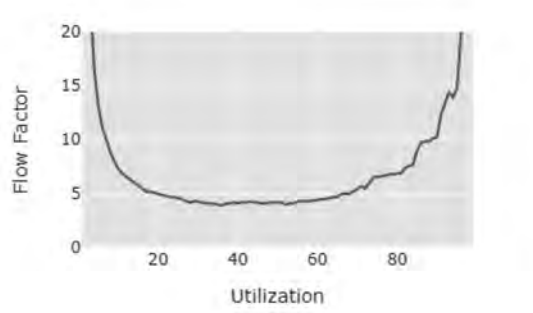

Figure 2: Operating Curve of a single batch-equipment with infrequent but long breakdowns

Examples of operating curves are shown in Figures 1 and 2. Although these curves represent the behavior of the system at all utilization levels, usually not the whole operating curve is relevant to a planner. What typically is of interest to our collegues is whether there is enough capacity for a given product mix or load. In modern days, this question has changed to whether it is possible to maintain a given flow factor with the given product mixes and loads. Therefore, it is important to know until which point an equipment group can be utilized before it violates flow factor targets. These thresholds are basi-

ARGESIM Report 59 (ISBN 978-3-901608-93-3), p 113-120, DOI: 10.11128/arep.59.a59016

cally what we are looking for. Figure 1 and Figure 2 also show that whether a system can handle a given material flow is not just based on its' utilization. Numerous factors are influencing equipment behavior; batching, breakdowns, and maintenance are just some examples. Based on these characteristics equipment groups are able to handle different utilization levels before reaching certain flow factors. As this differs from equipment to equipment this question needs to be answered often for different equipment groups.

It is the goal of our research to develop an approach that answers this question in a most timly fashion while still being sufficiently accurate to base planning and investment decisions upon. Traditionally, this is done at our industry partner with a calculation based on look up tables, which only included some factors. Although the look up is quite quick, the results were far from optimal since too few influencing factors were considered. A typical solution approach would simply be to build or generate a simulation model for a given equipment group and run some simuation experiments. But this would still take some time, with large equipment groups maybe even a few minutes. Hence, this approach would not meet the response time requirement for the given problem. Byrne [4] proposed an approach to limit the number of necessary design points to calculate an operating curve. Althougth this would speed up simulation, there would still be some waiting time for results.

In the first section of this paper, we will give an introduction to the general idea behind our approach. In Section two, we will discuss some software development aspects of creating such a system. In the third section, we will briefly show the features considered in our current project. In the following sections, we will furthermore discuss the simulation model, data farming, and training of our artificial neural networks. In Section seven, we will shortly show our current results and give an outlook on our future plans in Section eight.

1 System overview

As we have previously discussed, we aim to build a system which is able to quickly provide answers to the same question with changing parameters or configurations, that is repeatedly asked during a normal workday. If these questions would only be asked from time to time or a response would not be that time critical a common approach would be to build a simulation model and answer the question after analyzing a simulation experiment.

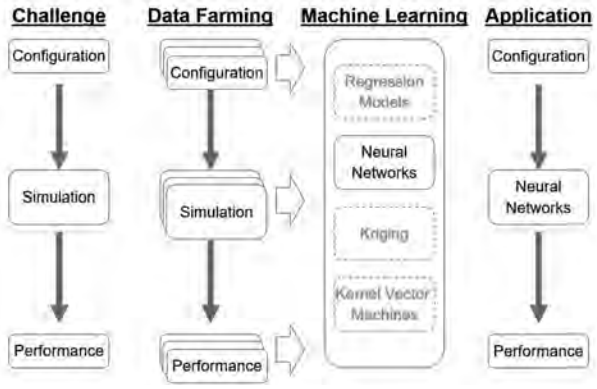

Figure 3: Generalization of discussed approach for a fast response system based on data farming and machine learning

If short response times are very important, Figure 3 shows a generalized approach on how we answer this question. The challenge column would be the normal simulation experiment approach. The user has a question about a given system configuration. A simulation model representing this system configuration is built and its' performance is evaluated. After a reasonable number of simulation runs the user gets the answer.

As the simulation runs are the time consuming part we changed the system. Instead of creating or parameterizing a simulation model each time the user needs an answer to the question we move the simulation runs to a point in time long before the user asks our system. We use data farming to create the results to a huge number of possible factor combinations. The resulting data set is then used as the supporting points for machine learing algorithms, in our case neural networks, to approximate a function that reproduces a response to a given configuration and therby replaces the simulation in the moment the user queries the system. Instead of directly asking for the results of a time-consuming simulation experiment, the user asks the neural network that is able to respond almost immediately.

Figure 4 shows a basic overview of our system.

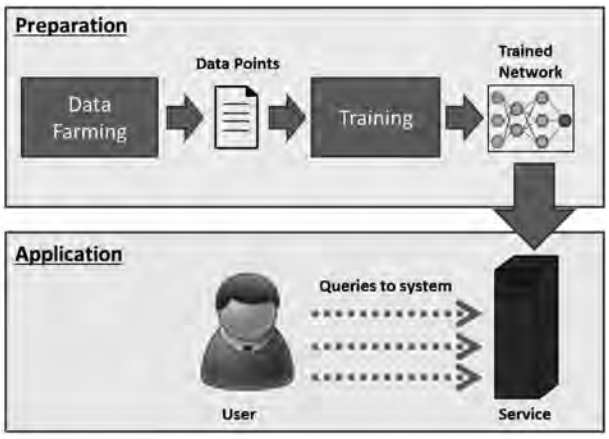

Figure 4: System overview

2 System architecture

In this section, we will discuss our system architecture from a software development point of view. We will start with the initial basic design and point our changes we have done to improve the system.

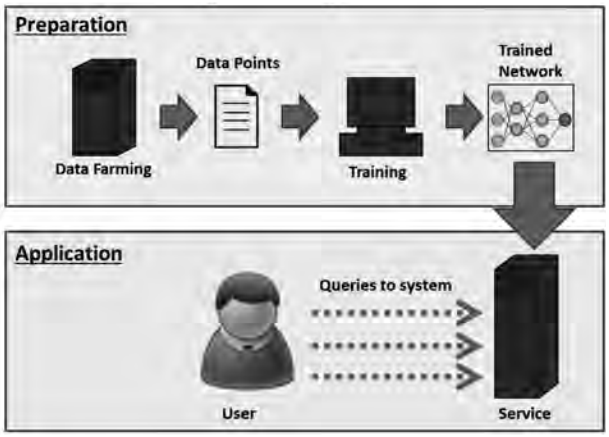

Figure 5: Basic system architecture

Basing our architecture (cf. Figure 5) on the system overview shown in Figure 4 we planned for one big simulation-based data farming component, which would generate a huge set of supporting points. These data points would be transferred as a file to an R (cf. [5]) script handling data preparation and training of the neural network. With this setup, we were able to obtain reasonable results but we found that there is still a lot of room for improvement. One of our first and surprisingly valuable changes was a switch from using R to train our neural networks to Keras (cf. [6]). With R, depending on our data set, we were sometimes observing training times of a couple of days, which we attributed to reaching some memory boundaries. We often had to abort after some time as no further progress was visible and it was hard to predict the remaining training time. However, even training times of a few hours considerably limit the amount of network configurations one can test in the hopes of improving results. The switch to Keras with an underlying Tensorflow (cf. [7]) library immediately improved training times incredibly. Furthermore, being able to utilize GPUs (graphics cards) for training improved training speeds to a point where instead of several hours or days we were looking at seconds and minutes for training. This new dimension of training times opened up the opportunity to consider neural architecture search to further improve the results of the neural network and thereby the whole system in the future.

A second big change to our system is the move from a monolithic piece of software to a service-based architecture using RESTful Web Services (cf. [8]). In this change, we see three major benefits to our system:

1. Ease of communication between system parts,
2. Scalability and distribution on multiple machines,
3. Replaceability of components.

In the beginning of the project, we made the conscious decision to implement different parts of our system with different languages. We see benefits in developing in Java with its object-oriented concept and type system paired with available IDEs supporting numerous ways of testing and debugging that make it very suitable to larger and more complex software projects. R on the other hand offered much easier access to mathematical functionality and neural networks. Nowadays, Python basically is the de facto standard language for data analyses and machine learning with a number of libraries and frameworks available and new systems usually being accessible only or at least first with Python.

Communication across these language barriers is often not easy with “direct calls”. As most modern languages nowadays offer libraries, to easily implement web services, this is an elegant approach to handle communication between system parts written in different programming languages without much additional implementation overhead. Gone were complicated command line calls and file-based communications.

Most parts in our system can be quite computation intensive. While it is still reasonable to run small test cases on a single office PC, larger experiments benefit from good scalability and distribution. Furthermore, different parts of the system benefit differently from

available hardware. While the simulation is mostly CPU and memory intensive, training neural networks significantly benefits from the availability of modern graphics cards. Being able to assign services to machines with the best fitting hardware is therefore another aspect, which is easily taken care of with services. Besides distributing the different services on different machines, some services may in the future need additional computing power. With a service-oriented architecture, it is also easy to introduce load balancers which distribute requests of the same type on several machines offering the same service without any necessary changes to the client side. This makes setting up such a system very comfortable for different experimental environments.

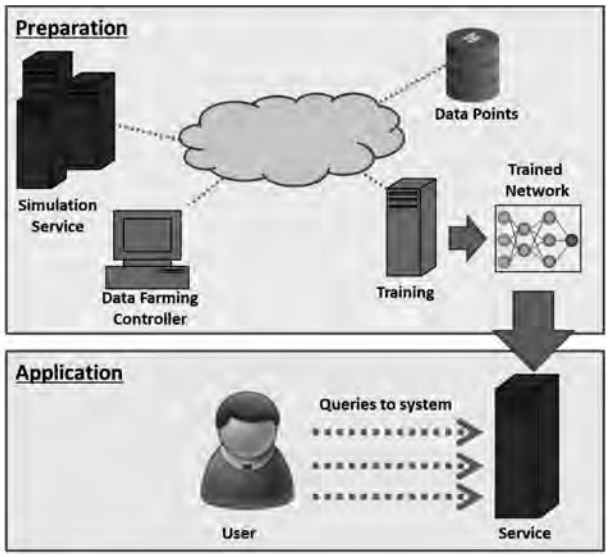

Figure 6: Service based system architecture

Replaceability without the need to touch any other system component is also a great benefit. As we try different frameworks and approaches the current architecture offers us to simply replace some services while others stay the same. Changing the simulator, the persistence approach or even the machine learning technique are all simply done by putting the new component up as a service replacing the old one. Hence, with this change we gained a lot of scalability and flexibility for future experiments.

3 Factors

Starting with Robinson et al. [1] and Hopp and Spearmean [9] and a review of the previous planning methods we defined relevant features for our equipment group model. Values for our factor levels were chosen based on a fab dataset from our industrial partner by looking for natural clusters and using representatives thereby capturing realistic workings points.

Most factors can be easily defined with single numerical values. These are shown as quantitative factors. Some factors shown as categorical in Table 1 represent more complex definitions. Product mix for example represents the number of different products as well as their percentage of the released material flow. For categorical features we selected three levels based on real equipment groups going from a low impact to a high impact setting with regard to the resulting flow factors.

Feature	Factor	#	Type
Batching	MaxBatch	5	Quant.
	MinBatchPercentage	3	Quant.
Breakdown	MeanTimeBetweenFailure	3	Quant.
	BreakdownCapaLoss%	2	Quant.
Dedication	Dedication	3	Cat.
Equipment #	ToolCount	7	Quant.
Maintenance	TimeToMaintenance	3	Quant.
	MaintCapaLos%	2	Quant.
Product Mix	ProductMix	3	Cat.
Rework	ReworkPercentage	3	Quant.
Process Time	RPT	6	Quant.
Setup	SetupDuration	3	Quant.

Table 1: Feature and factor overview

While initially considering only one factor per feature we now split some features into two factors for better scalability. This makes it easier for a future algorithm to generate new test points to validate and improve the resulting model. Additionally, the effect of some features is hard to capture with a single factor. For example, when considering two systems suffering from 25% loss of capacity due to breakdowns; one breaking down after 3 hours of productive time for about an hour while the other one runs fine for 3 weeks followed by a week of repair. We would expect the second system to perform worse with regard to cycle time and flow factor. Hence, we split breakdown into capacity loss and mean time between failures. As the increase in factors brings a significant increase in design points considering a full design, we have not yet updated our training data set to include all new points. Although the training data set does not include all these points, we are still able to address them better and test for them.

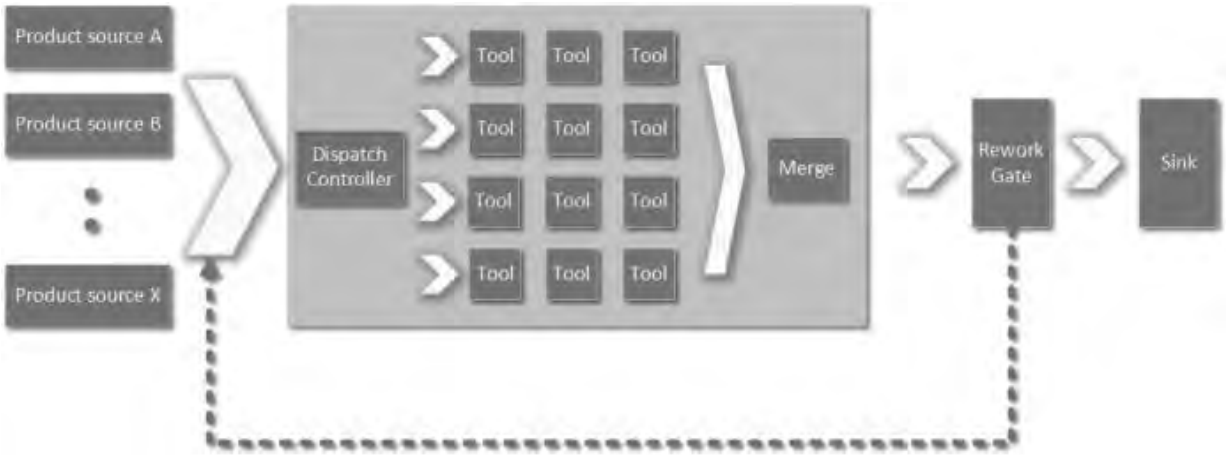

Figure 7: Simulation model structure

4 Evaluation and Simulation

We use an inhouse developed factory simulator for all simulation runs in this project. As we have mentioned before the simulator is currently running as a service and simulation runs can be started by calling the service and handing over parameter values for each factor under consideration.

The simulation service will then automatically generate a simulation model based on the given parameters. Figure 7 shows a visualization of the equipment group model. As the goal of the simulation is to determine reasonable utilization values for the lowest flow factor possible and the location of defined flow factor thresholds, the next step is a static capacity analysis. This is necessary to run the simulation only with reasonable loading scenarios without wasting calculation time for extremely low utilization settings or incredibly overloaded systems. Based on the static capacity analysis we can now simply calculate the necessary lot releases to run the model at a specific utilization point.

We use a search strategy akin to binary search to look for the location with the lowest possible flowfactor. For each utilization point under evaluation simulation runs are performed until the sample size for this point is determined to be large enough for a stable estimation. Flow factor thresholds are searched for similarly while reusing the results of previously tested plus new utilization points. Once the lowest flow factor value and all requested flow factor thresholds are determined the results are handed back.

Verification and validation are difficult when considering data farming, as it is almost impossible to evaluate every single simulation run. We deployed different strategies to ensure our simulation results reflect real world behaviour. The basis for this were unit tests to continuously check the simulation software during development. This was done to avoid unintended effects during programming. We additionally compared sample simulation results with the results from other simulation software packages. Additionally, we had a panel of experts reviewing results generated by the simulator and compare them to real factory data of equipment groups with similar characteristics. Of course, this cannot be done for all data points but helps to validate the system.

5 Data farming

When considering the number of factors and factor levels, we are looking at a huge number of data points to evaluate. In addition to all these data points, we are also looking at several simulation runs per data point. As we have mentioned before each data point is determined by calculating several utilization points for which we run a number of simulations each. Not all utilization points take the same amount of repetitions as we determine this number on the fly during the evaluation. After simulating an initial set of replications, we calculate the confidence interval half-length and mean. Then, we compare their quotient with the relative error we aim for. If the quotient is still larger than the relative error, we run another set of replications. We repeat this until the relative error is smaller than the quotient (cf. [10]).

On average, we ran about 825 simulations for a single data point to determine the location of the lowest flow factor value and three thresholds. Considering even just one factor per feature we were looking at almost 460000 data points which total in almost 380 million simulation runs just to generate the supporting points for

our project.

Although the evaluation of single data points is feasible on a normal PC, running these almost 460000 of data points on one of our simulation servers took several weeks. With our change in architecture and therefore much better scalability, we hope any future additions to our current data set will be available much faster.

6 Training

With all these data points from simulation, we were still only looking at the supporting points for our system. As we have mentioned before we moved from using R to Keras to implement the training of our neural networks which considerably increased training speed and made it much easier to test different layer configurations. Typically, we aim to minimize the mean squared error (MSE) of our testset. When trying to evaluate the usefulness of any trained network we additionally present the predicted results graphically.

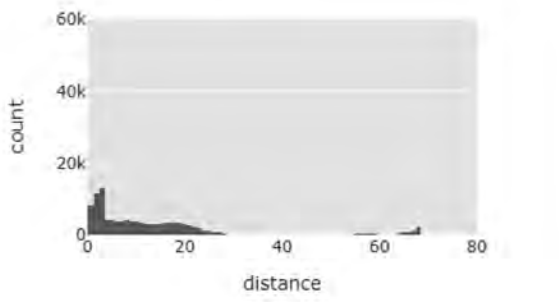

Figure 8: Visualization of the distance between predicted and simulated value; example 1

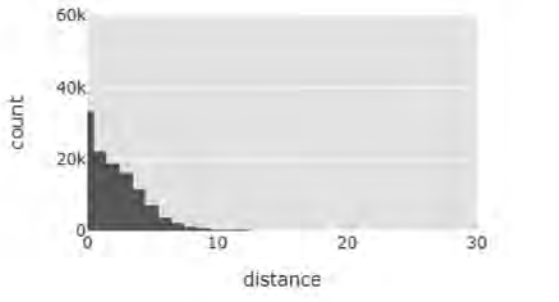

Figure 9: Visualization of the distance between predicted and simulated value; example 2

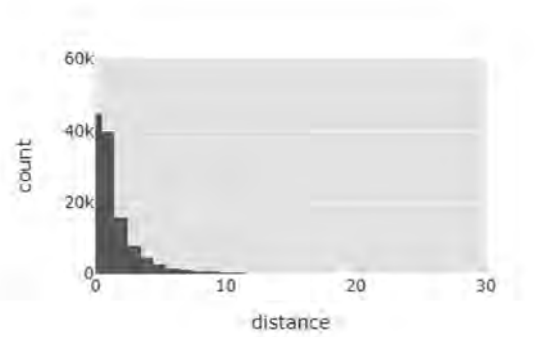

Figure 10: Visualization of the distance between predicted and simulated value; example 3

Figures 8 to 10 show the results of network configuration and training parameter sets we tested. The diagrams are historgrams of how successful the predictions have been. Starting from the left results are grouped by the error in prediction compared to the simulation result. The first bar represents less than 1% distance and each following bar an additional 1%. E.g., a scenario for which the network predicted 75.5 but the simulation estimated 73 would fall into the third bin. Please be aware that the x-axis in Figure 8 is using a different range from the other shown results. We chose to do this to be able to show the set of extremly poor predictions that is not present in the other results.

Although the quality of the trained networks can differ greatly between network configurations, all of the results shown here were able to reduce the MSE continuously during training and on a first glance seemed to work quite well. Only when visualizing what the results meant with regard to the actual problem at hand, it became obvious that some of these networks are not usefull at all to solve our problem.

Besides network architecture we found that training parameters like batch sizes and the number of episodes have a significant impact on result quality. In fact, Figures 8 and 9 are based on the same network configuretion but used different batch parameters for training.

7 Results

We set out to achieve two objectives. First, we wanted to create a system, which is able to respond immediately to a user query. Second, we needed to have sufficiently good results to base planning and investment decisions

upon.

On the first objective, we are where we want to be. The application server running on a better office PC with a modern graphics card responds within 300ms to a user query. The majority of this time is actually spent on allocating the graphics card. Running the network for the prediction just on the CPU without GPU support this response time could actually be reduced even more as predicting is quite fast with just the CPU.

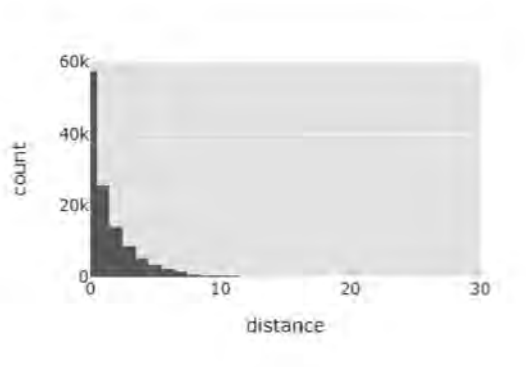

Figure 11: Visualization of the distance between predicted and simulated value; current results

Looking at the second objective, Figure 11 shows our currently best results derived just with manual testing of different network architectures and training parameters.
About 80% of our test points are predicted with an error of less than 3%. Almost all datapoints are predicted with less then 10% error from our simulation results. Furthermore, we are no longer seeing any artifacts as in Figure 8. Althoug these are reasonably good results and a prediction quality of 3% or better for a majority of data points would be good enough to base planning on these numbers, we are still looking at 20% of predictions being off by up to and 10%. Considering high utilization scenarios overestimating possible utilization by 10% error could have a serious impact on the performance of the material flow and ability to maintain promised delivery dates. On the other hand, underestimating utilization thresholds by 10% would mean significant loss of production capacity or triggering an investment in equipment before it is actually necessary. We therefore still see some need for improvement.

8 Summary and Outlook

In this paper, we presented our approach to create a system with minimal response time to a user query, which we would usually answer by simulation. We discussed some aspects of software architecture to improve scalability and flexibility of the software. The presented system uses a simulation model to do data farming for supporting points, which are then presented as training data set to machine learning methods like neural networks. The resulting trained system is able to respond to the user queries within moments.

Although the system already works as a proof of concept, the accuracy is still not where it would need to be to be applicable in an industrial setting. We are working on two approaches to improve prediction quality for all points.

First, we have seen during our manual configuration of the the neural network that network architecture and training parameters tend to have a big impact on result quality. With our improved training speeds, we are planning to improve prediction quality by automating the process of finding a good network configuration. There are several promising approaches to do this. We are currently working on adding a neuroevolution (A broader explanation can be found in [11]) service to our system. As an alternative, we are also looking at Auto-Keras (cf. [12]).

The second approach targets the somewhat infrequent supporting points within our training data set. Since adding additional levels to factors drastically increase the number of total points to calculate for a full design, simply adding more levels would be a very computation intensive approach. Instead, we are seeking to improve the quality of the systems response by automatically searching for points with bad predictions and adding additional supporting points near those points to our training data set. Idealy, this would improve prediction quality in those areas and therefore for the whole system.

Ackowledgements

We would like to thank Dr Thomas Mayer for many interesting discussions on system architecture and machine learning.

References

[1] Robinson, J.K., Fowler, J., Neacy E. *Capacity Loss Factors in Semiconductor Manufacturing*. FabTime Inc 2003

https://www.fabtime.com/files/CapPlan.pdf.

[2] Aurand, S., Miller, P. The operating curve: a method to measure and benchmark manufacturing line productivity. 1997 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop ASMC 97 Proceedings. *1997 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*; 1997 Sep; Cambridge, MA, USA. IEEE. 391-397. doi: 10.1109/ASMC.1997.630768.

[3] Fayed, A, Dunnigan B. Characterizing the Operating Curve — how can semiconductor fabs grade themselves?.. *2007 International Symposium on Semiconductor Manufacturing*; 2007 Oct; Santa Clara, CA, USA. Place of Publication: publisher. 1-4. doi: 10.1109/ISSM.2007.4446827.

[4] Byrne, N.M. *A framework for generating operational characteristic curves for semiconductor manufacturing systems using flexible and reusable discrete event simulations* [dissertation]. School of Mechanical and Manufacturing Engineering. Dublin City University; 2012.

[5] Verzani, J. *Using R for Introductory Statistics*. 2nd Edition. New York, USA: CRC Press; 2014.518p

[6] Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd Edition. Sebastopol: O'Reilly; 2019. 600p

[7] Abadi, M., Barham, P. et.al. TensorFlow: A System for Large-Scale Machine Learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*; 2016 Nov; Savannah, GA, USA. 265-283. ISBN: 978-1-931971-33-1.

[8] Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures* [dissertation]. University of California, Irvine; 2000.

[9] Hopp W.J., Spearman, M.L.. *Factory Physics*. 3rd Edition. New York: McGraw-Hill; 2008.

[10] Law, A.M., Kelton,W.D. *Simulation Modeling and Analysis*. 3rd Edition. New York: McGraw-Hill; 2000. 760p

[11] Stanley, K.O., Clune, J., Lehman, J. et al.. Designing neural networks through neuroevolution. *Nat Mach Intell.* 2019; 1: 24-35. doi: 10.1038/s42256-018-0006-z.

[12] Jin, H., Song, Q.,Hu, X. Auto-Keras: An Efficient Neural Architecture Search System. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*; 2019; Anchorage, AK, USA, NY, USA: Association for Computing Machinery. 1946-1956. doi: 10.1145/3292500.3330648.