Machine Learning and Neural Networks for Real-Time Scheduling

Adam Loree and Christy Wilhite
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816-2362

***Abstract*— Using neural networks to find optimal solutions to real-time scheduling is a common technique, and there have been many different models put forth to accomplish this goal. This paper is an academic literature review of six different designs put forth that use neural networks for real-time scheduling. A comparison is done for these models which weighs the feasibility and time complexity for each one as well as identifying common themes and trends in this topic.**

Keywords—Real-time systems, Machine Learning, real-time scheduling.

I. Introduction

Real-time systems are used widely in today's modern society. In addition to being used in personal computers as one would expect, they are used in mobile communications, automobiles, traffic management, and aeronautic systems just to name a few. Soft real-time systems allow tasks to finish past their deadlines and handle systems overloads. As pointed out by Guo and Baruah [1], this soft-time scheduling is frequently used to support quality of service (QoS) in applications such as video games, multimedia systems and telecommunication networks. Hard real-time scheduling is different as it considers worst case scenarios and schedules everything to handle those scenarios. These types of systems are used in things like self-driving cars, aerospace applications, and life-saving equipment that cannot tolerate any tasks being late or being dropped [1]. One infamous instance of a hard real-time system failing is the case of the 1997 Mars Rover Pathfinder. Priority inversion caused the system to keep resetting itself. Fortunately the problem was able to be solved by applying a priority inheritance protocol that NASA was able to load from earth, and the rover was saved [8].

Machine Learning (ML) is a subset of Artificial Intelligence that utilizes Neural Networks. Models are built using artificial neurons and then trained using large data sets. Once trained, the model can then make predictions for previously unseen data [7]. Researchers in the field of real-time scheduling have developed models to utilize ML to optimize systems and handle dynamic changes. Our research focuses on reviewing and comparing some of these implementations.

A. The Literature

The first paper reviewed was titled A Neurodynamic Approach for Real-Time Scheduling via Maximizing Piecewise Linear Utility by Guo and Baruah [1]. This paper discusses uniprocessor mixed-criticality scheduling problems where each job J_i is described by its release time a_i, a worst case execution time (WCET) estimation c_i, and a utility function μi (t) : [0,+∞) → R. Their focus is to maximize the overall utility of the scheduled jobs, but they consider separate utility functions for each job instead of a single function for the entire job set. They note that prior neurodynamic approaches have used recurrent neural networks (RNNs) to try to solve these scheduling problems, but they often have difficulty since these problems have been shown to be NP-hard and the global minima can be difficult to reach with gradient information. By using the concave linear piecewise function, they were able to solve the scheduling problems in polynomial time [1].

The next paper we did a deep dive into was Reducing network and computation complexities in neural based real-time scheduling scheme by Ruey-Maw Chen [6]. This paper did not take long to start the discussion of two-dimensional Hopfield-type neural networks using competitive rule. Before reading this paper we were not confident with this topic so it was difficult for us to accurately review this paper. After reading through most of it we were able to put together a very thorough understanding of its consequences in real time systems. Most importantly for solving three-dimensional multiprocessor real-time scheduling problems. As stated in this paper, using a two-dimensional network can lower the initial number of network neurons. This is important in lowering the computational complexity of these problems. Using three-dimensional networks would be detrimental to run time but using this proposed solution helps to greatly reduce the cost.

Reading Carlos Cardeira and Zoubir Mammeri's Neural Networks for Multiprocessor Real-Time Scheduling [5] was a good paper to review next because it describes a type of evolution to the Hopfield type neural network. This paper uses a type of evolved Hopfield network to analyze real time

scheduling problems and analyze the computational complexity and convergence rate.

Moving on to A Framework to Design and Implement Real-time Multicore Schedulers using Machine Learning [4]. This paper is very interesting and informative because instead of spitting facts at the reader they educate the reader on the topic. They take the time to describe the framework in detail and then demonstrate its applications in real time systems. They begin by introducing their framework to design and implement multicore schedulers in real-time using machine learning. The framework captures run time data and subjects it to the ML tools to create targeted optimization goals. It also utilizes the performance monitoring unit, thermal sensing, energy monitoring, and dynamic voltage and frequency scaling through an API [4].

The paper titled Solving Real-Time Scheduling Problems with Hopfield-Type Neural Networks by Silva et al. is an older paper that was written in 1997, and it discusses the basics of Hopfield-type neural networks and how they can be used in real-time scheduling [2]. They note that it consists of a large number of neurons whose output is binary in nature and can converge very quickly. The researchers present a systematic method of mapping preemptable task sets onto a multiprocessor system resulting in schedules that are always feasible, and "global asymptotic consistency between the discrete time model and the continuous one is assured". Figure 1 is their representation of a single neuron in the Hopfield neural network [2].

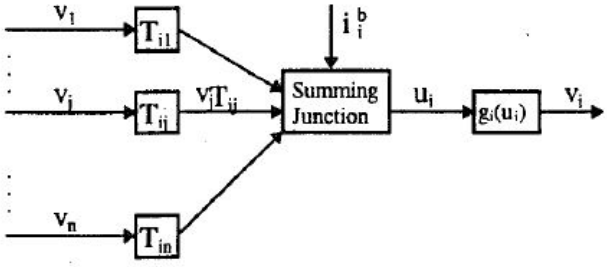

Figure 1: Additive model of a neuron [2]

The last paper to be reviewed was Scheduling Multiprocessor Job with Resource and Timing Constraints Using Neural Networks by Huang and Chen [3]. The design put forth in this paper is also built on a Hopfield neural network with multiprocessors. The researchers build on the technique stochastic simulated annealing to come up and apply a newer method called mean field annealing which uses the mean field approximation technique. Their work contends that this method of normalizing simulated annealing can efficiently solve combinatorial optimization problems such as task scheduling. Figure 2 shows their representation of the 3D Hopfield neural network which is discussed extensively throughout this literature review [3].

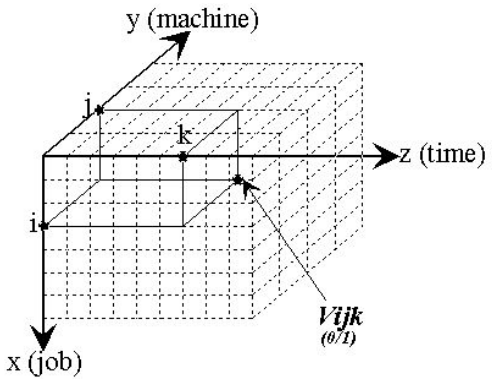

Fig. 2: 3D Hopfield Neural Network [3]

II. Design Implementations

This section will examine the proposed design implementations described in these papers and how they perform in more detail.

In the first paper [1], Guo and Baruah propose a method that computes a utility function for each job as it is executed to maximize the overall utility of the task set. The problem of finding optimal utility for a real-time task set has been proven to be NP-hard, and can be represented by a step function. However, by replacing the nonconvex step function with piecewise linear concave object function, the problem is transformed into a constrained convex optimization problem able to be solved in polynomial time. The neurodynamic approach involves vectorizing a point in the RNN system and then performing a transformation to come up with a scheduling matrix. After applying the RNN, the state variable is recorded, and the matrix is transformed back into the scheduling table. Since the RNN is globally convergent, initial states do not matter so they are set to zero [1].

Guo and Baruah [1] tested their model extensively which included overloaded examples. They note that during the RNN convergence process, the convergence time did not suffer even though the number of jobs was increasing. This could indicate that this model may be useful in large scale applications such as cloud computing. In their example problem of five jobs, two of the less important jobs were dropped, and the task set ended up having a total utility of 1.249. When compared with traditional EDF, all jobs are executed but only 2 jobs met their deadlines, and the total utility was significantly less at 0.623. They point out that missing deadlines in hard real-time systems is just as bad, if not worse, than dropping the job instead. When tested on job sets that were not overloaded, the model performed optimally [1].

Guo and Baruah [1] further note that their model has only been tested on uniprocessor systems, and future research into using it in a multiprocessor system may be beneficial. They also did not include any penalty for preemptions in their

calculations, so an additional term could be investigated that would add this penalty to the calculations [1].

Reducing network and computation complexities in neural based real-time scheduling scheme by Ruey-Maw Chen [6] starts out by talking about commonly known real time problems and how they can be solved with different types of algorithms. They go into detail and talk about the implications on real life problems. These problems are sometimes built into three-dimensions or more neural networks and this paper aims at minimizing computational time while maximizing efficiency. The specific study of the computational complexity is important and it is heavily discussed in relation to two-dimensional and three-dimensional neural networks. Within a three-dimensional network the amount of neurons is said to be (N x M x T) and the proposed two-dimensional network would only consist of (N x T) neurons. Consequently this would make the upper bound of the computational complexity $O(N^2 \times T^2)$. Using this computational complexity we know that the scheduling algorithm may need to invest [6] significant resources to determine synaptic weight. This is shown to be $O((N \times T)(N + T))$. The added complexity of (N + T) is equivalent to the algorithmic operations needed to calculate the synaptic weight.

Conclusively turning a three-dimensional neural network into a two-dimensional M-out-of-N competitive scheme results in a simplified approach. Using this approach like we said reduces the computational complexity of this problem from $O(M \times N \times T)$ to $O(N^2 \times T^2)$ not considering other iterations and synaptic weights [6].

Much of current real time systems research has been carried out in real world applications as described in this paper [5], like speech recognition and image compression. A lot of recognition has real time system implications. In this paper's references to previous work on neural networks and real time systems they mention Liu and Leyland which have been very prominent people in my real time systems education. They also talk about the K-out-of-N rule. This rule is an important rule in building neural networks. To satisfy the constraints of this rule there must be exactly K neurons among N activated when the network reaches the stable state. The outlines of the actual equation are in this paper. Then they go on to use this rule in their translating constraints into rules section. They use this rule when analyzing optimization problems with neural networks. Using the K-out-of-N rule to translate real-time tasks into an energy function is done by using a network topology with a number of neurons equal to time units multiplied by the number of tasks. Described as L x T respectively. Next rule described is the P-out-of-T rule which is used when the tasks fully utilize the processors. The number of activated neurons must be the same as the number of processors which is why the P-out-of-T rule is required.

Creating an example of this functionality is done in this paper [5] by simulating a neural network and tracking activated and not activated neurons. The purpose of testing is to see if the neural network will evolve to solve the real time problem. In this task there are 3 processors and 6 time units which will give us 18 neurons as described earlier (L x T). In this example the neural network is built with successive passes of the K-out-of-N rule. The final complexity analysis is done by analyzing the data plotted on a graph. They analyze the number of tasks and the mean number of iteration taken by the chosen algorithm to converge to a solution. Eventually they decide the algorithm exists with an upper bound of $O(T^2 \times L^2)$ complexity. They also briefly mention that the complexity could be reduced to (T + L) if the fact that neurons are only connected to the same line and column.

This paper [5] also briefly tasks about the downfall of using their algorithm and neural networks in real time systems which we think is worth mentioning. They talk about how there may be special hardware to use neural nets like parallel hardware architectures. These special types of equipment may not always be available in a real-time system. They also mention that the network has a possibility of falling back into the local minima of the energy function [5] essentially ignoring the efficient and useful parts of the algorithm.

The architecture of this framework [4] is a queue manager that keeps the tasks order by criteria given while accessing a resource. The initial idea is given through a basic framework which can be flexible and become many different things. Real time threads in this architecture are simulated using endowing aperiodic threads. They use semaphoric synchronization mechanisms. When a thread is initialized the semaphores and synchronization mechanics are also initialized. The semaphores are initialized at 0. What is really interesting is the monitoring of this architecture. The structure of the architecture is said to be able to implement virtually any scheduling policy. One exception are algorithms that collect data at run time which dynamically adjusts the ordering of objects in scheduling queues. This is where the monitor class comes in. The monitor class coordinates with the thread class to create a run-time monitoring system. This system can collect data from different types of sensors available to the system.

The purpose of data collection when it comes to monitoring systems is to identify patterns and be able to replicate them. The monitoring system is able to ruin these patterns so it is important that the monitoring is non-intrusive. Therefore to prevent this limitations are put on sensors to prevent the number of access or even bandwidth accessed. Another way this is controlled is by using SmartData. SmartData is data that was captured on the monitoring system and subsequently converted to SmartData. The SmartData is described as data enriched with metadata to make it self-contained in respect to semantics, spatial location, timing and trustfulness [4]. This SmartData can be used in multiple different ways to diagnose and analyze the system. The overall goal of collecting this data is to be able to learn from it.

In the paper by Silva et al. [2], a "systematic procedure to map task scheduling problems onto Hopfield-type neural networks" is provided. Their algorithm handles periodic tasks but can take aperiodic ones and change them into periodic tasks in order to handle those also. Their design relies on multiprocessors that have identical processor speeds. They allow preemptions, and precedence and resource constraints are not considered in their calculations [2].

As previously discussed, Hopfield neural networks have binary outputs, so the problem solutions must be presented in a way that can be solved with a binary result [2]. To do so, they create matrix V that is the total number of jobs by the total number of time units (t x u). The number of columns are equal to the hyperperiod which is the least common multiple (LCM) of all task periods. They note that when viewing a task scheduling problem as an optimization problem, there is a criterion that you would be trying to minimize. However, in hard real-time scheduling it is more important to guarantee that the important tasks will be executed. The authors provide a set of equations to represent the problem as a quadratic 0-1 optimization problem. Their algorithm makes it possible to represent the task set with linear equality and inequality constraints. Timing constraints can be handled by the equality ones while hardware constraints can be handled by the inequality constraints. Functional constraints are not addressed here, but the authors note that they have covered that issue in one of their previous papers titled *Handling precedence constraints with neural network based real-time scheduling algorithms* which can be found in their references.

Silva et al. note that the mutual exclusion of the processors has to be handled with inequality constraints, but those must be converted into equality constraints [2]. They introduce a set of extended variables v and w which includes a slack variable. They go on to show examples of how the hardware constraint problems are computed with these vectors. Next they address the timing constraints which are equality constraints with instructions on setting up the vectors and showing example problems. They use a mapping technique based on work by Aiyer and Gee who are referenced in their paper. The method assures convergence and will verify the imposed constraints. However, this only works if all values are binary in nature, so they use annealing techniques to force the convergences to 0-1 points. They provide detailed results on how the network responds to simulations, and their evaluation determined that time complexity is dominated by $O(n^3)$ where n is equal to the number of neurons in the network [2].

In paper [3], Huang and Chen discuss applying a different annealing technique with the Hopfield neural network (HNN) to solve combinatorial problems such as real-time scheduling tasks. They provide a good deal of background on the Hopfield neural network which was developed by Hopfield and Tank in 1985. It's symmetrically interconnected network makes it effective at solving combinatorial problems such as the Traveling Salesman one. However, the binary output is not guaranteed to be optimal, so other researchers have looked into using simulated annealing (SA) to the network to address this problem. SA applies synaptic noise using thermal fluctuations which prohibits getting stuck at local minima and allows the global minimum to be reached. However, the HNN neurons are governed by the Boltzmann state-transition rule which means that a state change is only acceptable when there is an energy decrease. This implies that the HNN corresponds to a noiseless system while SA creates noise. Instead the authors chose to apply the mean field annealing (MFA) technique instead of SA as an alternative that better supports the HNN system [3].

The Huang and Chen [3] model proposed is based on a multiprocessor system that does not allow migration of jobs or resources. The jobs are broken down into segments, and preemption can only occur in between those segments. They then developed an energy function which is based on constraints from term to term instead of using the k-out-of-N rule. The constraints considered are confining the output state to a steady representation and deadline/resource constraints. Their function considers job, machine, and time variables, and their paper can be referenced for detailed descriptions of their derivation. The function is used and "transformed into the corresponding neural network for utilizing the HNN and the normalized MFA algorithm". They provide a set of equations that expand the HNN into a 3D state in order to map their energy function to it [3].

Huang and Chen [3] then discuss the MFA algorithm which is used to keep the network close to thermal equilibrium. If a state change results in lower energy, then it will be accepted. But if it doesn't, then a probabilistic process must be used to deal with the energy increase. The normalized MFA algorithm that they use has three steps. First they set the initial average state and start with a high temperature. Then they go through their sequential iterations as described in their section 3B equations, and then they decrease the annealing temperature and go through the iterations again until they reach convergence [3].

In their simulation testing [3], Huang and Chen note that the HNN goes through an oscillation process before it is finally able to converge, but with the MFA there is a smooth and efficient process to reach convergence. They note that their research focuses on resource utilization but could be expanded to include the time constraints for required resources for each job. They determined from their simulations that the time complexity of their model is $O(N^2*M^2*T^2)$ which does not increase linearly when the number of jobs increases. They note that future work should try to reduce this time complexity [3].

III. Comparison and Discussion

In reading all of these papers we have learned so much about the different types of machine learning in real time system scenarios. We have experienced different types of

comparisons between ranging from different neural network types and architectures. What stands out from all of these different approaches are the different computational complexities. It is interesting that using different methods for solving real time systems with machine learning will give such different computational complexities. For instance, solving the three-dimensional multiprocessor real-time scheduling problems with two-dimensional competitive rule is interesting because it gives us a better runtime than simply using a three-dimensional scheme for it.

Table 1 compares the designs that were reviewed in this paper. It shows if the model is suitable for uni or multi processors, what type of neural network was used, and the time complexity of the model if it was given. While these models are difficult to compare directly because they have different constraints considered, this table can provide an overall view of them.

Table 1: Comparison of Models

Design	Processors	Type of NN	Time Complexity
[1]	uni	RNN	polynomial
[2]	multi	Hopfield	$O(n^3)$
[3]	multi	Hopfield	$O(N^2*M^2*T^2)$
[4]	multi	N/A	N/A
[5]	multi	Hopfield	$O(T^2 * L^2)$
[6]	multi	2D Hopfield	$O(N \times T^2)$

We also observed a strong trend in using the Hopfield neural networks for real-time scheduling, especially early on in the research. However, we see in a later paper that recurrent neural networks are also being used, and the comparison between the two could be a subject for another research project. None of these papers used a convolutional neural network, so it would be interesting to see if there are any implementations of that to be found.

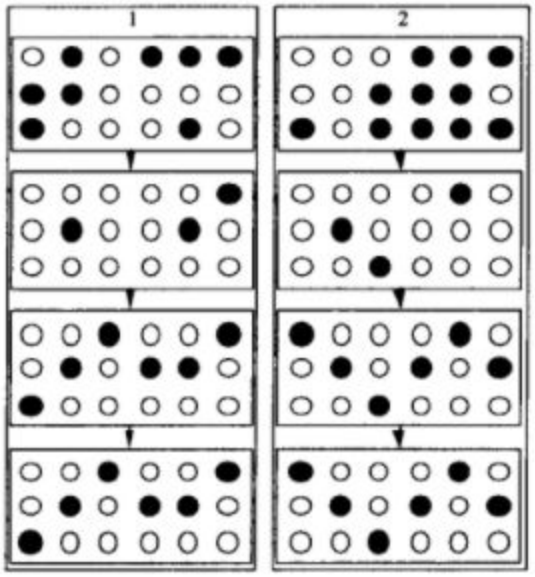

Fig. 3: Simulation of the neural net behaviour. [5]

Before coming into this project our understanding of neural networks and how they were built was limited. After observing the different types of building for these neural networks in many different types of papers it is easier to see how they are built and evolved. For instance, in figure 3 we can see the two different lines of evolution independent of each other. The dark neurons represent active neurons and the white ones represent inactive ones. In this paper [4] the evolution is designed by a c coded program that is a framework for designing neural networks that use machine learning to solve real time system problems. You can see in the figure how it converges in one to two evolutions.

IV. Conclusion

This literature review has covered six academic research papers on using Machine Learning neural networks to solve real-time scheduling problems. Many of the neural networks that were used in the models were Hopfield neural networks which we have discussed throughout this paper. However, there is also a recurrent neural network used in one of the newer papers which may indicate a trend in that direction. Some of the models used annealing techniques. Some change the neural network from 2D to 3D or from 3D to 2D. The implementations of these models is discussed, and the models are compared by their constraints and time complexities.

Ultimately the study of these neural networks have shown that there are many different ways to approach the same problem. Using different types of neural networks can yield many different types of results. What is more interesting is that the time complexity can different from design to design even when the neural network is the same. For instance, a Hopfield type neural network was used for half the papers we

reviewed. All the different approaches resulted in similar but unique computational complexities.

References

[1] Z. Guo, and S. K. Baruah. "A Neurodynamic Approach for Real-Time Scheduling via Maximizing Piecewise Linear Utility." IEEE Transactions on Neural Networks and Learning Systems, Neural Networks and Learning Systems, IEEE Transactions on, IEEE Trans. Neural Netw. Learning Syst , vol. 27, no. 2, pp. 238–248, Feb. 2016.

[2] M. P. Silva, C. Cardeira, and Z. Mammeri. "Solving Real-Time Scheduling Problems with Hopfield-Type Neural Networks." EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No.97TB100167), EUROMICRO 97. New Frontiers of Information Technology., Proceedings of the 23rd EUROMICRO Conference , pp. 671–678, Jan. 1997.

[3] Y. M. Huang and R. M. Chen, "Scheduling Multiprocessor Job with Resource and Timing Constraints Using Neural Networks", IEEE Transactions on systems, Man and Cybernetics-Part B , vol. 29, no. 4, pp. 490-502, 1999.

[4] L. P. Horstmann, J. L. C. Hoffmann, and A.A. Frohlich, "A Framework to Design and Implement Real-Time Multicore Schedulers using Machine Learning", 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) Emerging Technologies and Factory Automation (ETFA), pp. 251-258, Sep. 2019.

[5] Cardeira, C., and Z. Mammeri. "Neural Networks for Multiprocessor Real-Time Scheduling." Proceedings Sixth Euromicro Workshop on Real-Time Systems , Sixth Euromicro Workshop On Real-Time Systems , pp. 59–64, 1994.

[6] R. M. Chen, "Reducing network and computation complexities in neural based real-time scheduling scheme", Applied Mathematics and Computation vol. 217, no. 13, pp. 6379-6389, 2011.

[7] L. Wang, Class Lecture, Topic: "Artificial Intelligence- Introduction", CAP4630, College of Engineering & Computer Science, University of Central Florida, Aug. 27, 2020.

[8] Z. Guo, Class Lecture, Topic: "Real-Time Systems- Introduction", EEL4775, College of Engineering & Computer Science, University of Central Florida, Aug. 26, 2020.