Cheng-Hung Wu, Fang-Yi Zhou, Chi-Kang Tsai, Cheng-Juei Yu & Stéphane Dauzère-Pérès (2020) A deep learning approach for the dynamic dispatching of unreliable machines in re-entrant production systems, International Journal of Production Research, 58:9, 2822-2840, DOI: 10.1080/00207543.2020.1727041

# A Deep Learning Approach for the Dynamic Dispatching of Unreliable Machines in Re-entrant Production Systems


## Abstract

This research combines deep neural network (DNN) and Markov decision processes (MDP) for the dynamic dispatching of re-entrant production systems. In re-entrant production systems, jobs enter the same workstation multiple times and dynamic dispatching oftentimes aims to dynamically assign different priorities to various job groups to minimize weighted cycle time or maximize throughput. MDP is an effective tool for dynamic production control, but it suffers from two major challenges in dynamic control problems. First, the curse of dimensionality limits the computational performance of solving large MDP problems. Second, a different model should be built and solved after system configuration is changed. DNN is used to overcome both challenges by learning directly from optimal dispatching policies generated by MDP. Results suggest that a properly trained DNN model can instantly generate near-optimal dynamic control policies for large problems. The quality of the DNN solution is compared with the optimal dynamic control policies through the standard K-fold cross-validation test and discrete event simulation. On average, the performance of the DNN policy is within 2% of optimal in both tests. The proposed artificial intelligence algorithm illustrates the potential of machine learning methods in manufacturing applications.



**Keywords:** Deep Learning, Deep Neural Network, Dynamic Dispatching, Markov Decision Processes


# 1. Introduction

Recent manufacturing technologies enable the use of sensors on network-connected machines and make production control increasingly complex. Additional sensors allow more uncertain events to be monitored, and network connection capabilities allow machines to be remotely controlled in real time. Markov decision processes (MDP) are the most widely used tools for dynamic decision-making. Applying MDP to model additional status information and dynamically adjust manufacturing decisions provides opportunities for performance improvement. In a production environment, however, it remains unclear how the additional information could be efficiently modeled due to the following challenges.

1. Curse of dimensionality in large-scale MDP problems

The well-known curse of dimensionality of MDP refers to the fact that the computational time required to solve MDP models increases rapidly with the dimensions of state and action variables. In production systems, numerous sensors imply that many machine and system state variable dimensions are being monitored. Given that each state variable represents a dimension of the state space, the total number of elements in the state space quickly increases with the number of sensors. The dimension of action variables also increases because the number of real-time control decisions allowed in network-connected machines rises. The computational complexity of solving MDP models is proportional to the number of possible

combinations of state and action variables; thus, the required time for solving optimal dynamic dispatching problems increases exponentially with the number of machines when machine failures are considered, Wu et al. (2006).

2. Frequent need to rebuild MDP models: the barriers between dynamic programming models

The barriers between dynamic programming models also prevent the use of dynamic control in the production environment. The optimal control policy of an MDP model can be found by iteratively solving the optimality equation of the MDP model. The optimality equation depends on the state space, action space, transition probability between states, and state- and action-dependent cost functions. Given that the state space, action space, transition probability, and cost functions depend on system configuration, a new MDP model with a new optimality equation is required after every change in system configuration. For example, when a new machine is introduced, machine availability state changes accordingly. In another example, when a new product is introduced, differences among production processes change production rates. Building a different MDP model and solving this new model are always necessary after system parameter or configuration is changed. Given that short product life cycles and high product varieties are common nowadays, system configuration frequently changes. Such changes lead to the repeated building and solving of new MDP models. Experienced engineers and programmers are frequently required to modify computer codes to develop and implement new models due to the lack of general-purpose solvers for MDP problems. Therefore, the complexity of solving MDP models hinders the adoption of MDP in dynamic production control.

Although the use of optimal dynamic control in production systems considerably improves operational efficiency in research(e.g. Ahn et al. (1999)), realizing dynamic production control in practice remains unclear due to the curse of dimensionality and the barrier between MDP models. For example, a typical semiconductor fabrication plant (hereafter referred to as “semiconductor fab”) is a highly dynamic environment where configuration changes frequently. In the facility of an industry partner, product mix changes nearly every week and the number of available machines for each operation changes daily with machine qualification processes. Given that the computational time for solving MDP models increases exponentially with system size, solving the dispatching problem may take days or even weeks, which is computationally inefficient for large systems. Therefore, solving a new MDP model after every configuration change is infeasible in semiconductor manufacturing.

The objective of the current research is to combine MDP with deep learning (DL) to construct an efficient dynamic control policy generator for a variety of dynamic dispatching problems. The proposed deep neural network (DNN) approach is a particularly viable option in a dynamic production environment that is subject to frequent configuration changes. The training data of the proposed DNN model are the optimal control under hundreds of different system configurations. From the training data under different system configurations, the DNN learns how optimal control policies will vary with system configuration. Thus, when a new system configuration is introduced, the proposed DNN model can quickly predict new control policies and eliminate the need to frequently solve new MDP models. Accordingly, collecting new training data and retraining the

DNN are unnecessary after every configuration change. In our numerical study, the proposed method is validated in reentrant manufacturing systems, such as semiconductor manufacturing systems and the numerical results indicate that the performance of the DNN policy is within 2% of optimal on average.

# 2. Literature Review

To satisfy ever-changing customer needs, companies are currently producing various products and product life cycles are becoming shorter. The adoption of multifunctional machines is critical for fulfilling the varying production requirements of different products. On the one hand, the additional flexibility of multifunctional machines allows the flexible use of valuable capacity under uncertain demands. On the other hand, additional machine flexibility makes dynamic production control an emerging challenge for production systems. In the literature, researchers have developed dynamic production control methods through queuing system analysis and MDP to efficiently use the additional flexibility of machines. Gershwin (1987) and Chang and Gershwin (2010) evaluate the performance of serial finite buffer production systems with and without machine failures. Ahn et al. (1999) study a two-stage tandem queuing system with two multifunctional machines. They present an optimal control policy that is monotone to the amount of work in process (WIP) in a queue. Although machines are unreliable, Wu et al. (2006) prove the optimality of threshold-type dispatching policies in two-stage flexible production systems. Kirkizlar (2008) compares fully flexible with partially flexible machines and finds a similarity between their corresponding optimal control policies. Wu et al. (2010) develops efficient algorithm for the dynamic production control in tandem lines, where the waiting time of jobs subjects to time window constraints. Choi and Kim (2012) develop a neural-dynamic programming method for the scheduling problems in re-entrant production systems with limited capacity. All the aforementioned studies show an optimal dispatch policy that depends on the real-time queue length and reliability status of machines. Thus, the resulting optimal control decisions depend on the real-time status of a system.

When researchers attempted to solve large control problems through queuing and dynamic optimization methods, they started to encounter the curse of dimensionality of MDP. To address this phenomenon, an effective approach is to decompose a large system into several subsystems. Meyn (2005) finds that a deterministic workload for stochastic network models provides the upper and lower bounds for controlled Brownian motion. Archibald (2007) uses transshipment policies as a means to decompose large decision problems. Wu et al. (2008) propose a decomposition algorithm to break down serial production lines into several two-stage subsystems. For a sustainable hybrid flow shop, Shi et al. (2019) develop a genetic algorithm (GA) for multi-objective dynamic scheduling problems through the multi-agent approach. Xia et al. (2019) propose a decomposition method using generalized exponential distribution to analyze a transfer line. The decomposition method considers unreliable buffer with time-dependent failures.

However, solving dynamic production control problems in large systems remains computationally infeasible even with a decomposition algorithm (Puterman, 1994). Therefore, researchers have used neural networks to study the applicability of simple heuristic dispatching rules. Mouelhi-Chibani and Pierreval (2010)

propose a real-time dispatching method using an artificial neural network. Olafsson and Li (2010) study a learning method to choose optimal dispatching rules through a decision tree. Shiue (2009) uses support vector machines to develop a dynamic dispatching mechanism for shop floor control problems. Priore et al. (Priore, Parreno, Pino, Gomez, & Puente, 2010) propose a neural network approach to learn scheduling rules from simplified production systems. Zhang and Wang (2016) develop a multi-agent hierarchical heuristic for the scheduling of re-entrant manufacturing systems. For distributed manufacturing systems, a cloud-based framework allowing effective planning and scheduling is proposed in Mishra et al (2016). Instead of learning from optimal dynamic control policies, the aforementioned studies use heuristic approach and are bound to performance loss due to the limited information brought by simple heuristics.

Reinforcement learning (RL) is another machine learning approach for the dynamic control of large systems. RL, also known as approximate dynamic programming, starts by estimating the value functions of dynamic programming. In this area, many researchers have used neural networks as their supervised value function approximation tool. Marbach and Tsitsiklis (2001) propose an approximation of policies by Q-factors through neural networks, but their method may suffer from slow convergence. Tsitsiklis and Van Roy (1999) develop a linear approximation to fit value function instead of using a restriction-fixed policy. RL is also used in control problems. For example, Marbach, Mihatsch, and Tsitsiklis (2000) use a classic neural network as a simulator to solve the admission control problem in single-link service networks. Simester, Sun, and Tsitsiklis (2006) use RL to find the optimal mail order, and the model converges to the long-run optimal solution. Aydin (2000) presents an RL method for dynamic job-shop scheduling problems using the agent search method. Zhou et al. (Zhou, Wu, & Yu, 2017) combine RL and DNN in a dispatching problem, in which DNN is used to evaluate the value function. All the aforementioned studies on RL learn from the value function or the Q function of an MDP model. However, their limitation is that the learning process must be repeated after system configuration changes due to the barrier between MDP models. The same remodeling issue also exists in these RL studies; thus, modeling and solving a large dynamic optimization problem are frequently required.

While manufacturing efficiencies are oftentimes hampered by configuration changes and various uncertain factors, our research shows the potential of artificial intelligence methods in dynamic production control. To eliminate the need for frequent model solving, the major contribution of the current research is to develop a new artificial intelligence (AI)-based method that instantly generates new control policies after system configuration is changed. By combining DNN with MDP, we establish neural networks to learn from the optimal dynamic control policies generated by MDP. The relationship between optimal control policies and system configurations is characterized by the trained DNN. When a new system configuration is introduced, the trained DNN model becomes an effective predictor for the new optimal policy.

Compared with the existing literature, the major contributions of the current research are as follows.

1. The proposed AI method is effective in overcoming problems caused by the curse of dimensionality. In contrast with typical RL approaches that learn from value functions, the proposed DNN is formulated to learn directly from the control policy space that represents the optimal dynamic control of small systems. The DNN is demonstrated to be fast and effective in predicting optimal policies in large problems.

2. The proposed AI method is reusable after system configuration is changed. Thus, the AI method overcomes the drawback of traditional MDP methods that are required to build and solve new models after system parameters are changed.
3. Instead of learning from human decision makers, the proposed DNN learns directly from optimal control policies generated by dynamic programming. Thus, it prevents the possible bounded rationality of human decision makers. The DNN can generate near-optimal control policies in large systems. In our numerical validation, the average gap is less than 2% compared with that of optimal control policies.
4. The proposed AI approach can immediately generate dynamic production control policies and allow easy integration into existing manufacturing execution systems. Thus, high-quality dynamic control is possible in the production environment.

# 3. Problem and Model Formulation

In this research, MDP is used to generate optimal dynamic dispatching policies in a re-entrant workstation with dedicated and flexible machines. The objective of the MDP model is to minimize the long-run weighted average cycle time when machines are unreliable and subject to random failures. In the re-entrant line, all jobs visit the workstation twice to finish two different operations (Fig. 1).

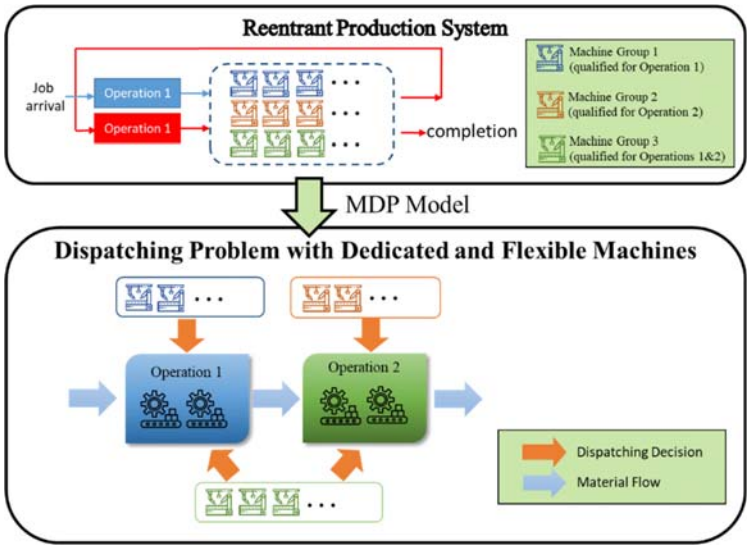



Fig. 1 Dispatching of flexible machines in a re-entrant system

Nattaf et al. (Nattaf, Dauzere-Peres, Yugma, & Wu, 2019) report that in semiconductor and other reentrant production systems, not every machine is qualified for all operations due to quality constraints and other process considerations. Thus, in our MDP model, several machines are dedicated to either the first or second operation, whereas other machines are flexible and can be dynamically allocated to process both operations. To minimize weighted average cycle time in dynamic dispatching problems, different priorities are assigned to upstream and downstream jobs depending on the real-time status of a system. In consideration of machine failures, dispatching decision not only depends on real-time queue length but also on machine health status. In the literature, MDP is an effective tool for the dynamic dispatching problem and can save average cycle time by up to 70% compared with several simple heuristics that are commonly used in production systems (C.-H. Wu et al., 2006).

Given that the MDP and DNN models require a clear definition of state and action variables, the MDP model should be defined first before we can introduce the proposed DNN approach. The major contribution of this research is the AI-based approach, and the MDP dispatching model is similar to that of Wu et al. (C.-H. Wu et al., 2006). Thus, although the MDP model is defined clearly in the next section, it is not discussed in detail. Interested readers may refer to Wu et al. (C.-H. Wu et al., 2006) for an in-depth discussion of the MDP dispatching model.

## 3.1 MDP model for the dynamic dispatching problem

System parameters are defined as a combination of pivotal features, such as the maximum number of parallel machines in each workstation, the service rate of machines, the reliability of machines, and the arrival rate of jobs. The notations used in our model are summarized in Table 1.

Table 1 Nomenclature

| Notation | Introduction |
|---|---|
| $i$ | $i \in \{1,2,3\}$, machine groups<br>$i = 1$, dedicated machines for upstream operation 1<br>$i = 2$, dedicated machines for downstream operation 2<br>$i = 3$, flexible machines for operations 1 and 2 |
| $j$ | $j \in \{1,2\}$, operations<br>$j = 1$, upstream operation 1<br>$j = 2$, downstream operation 2 |
| $T$ | Planning horizon $T \in \{1,2,\dots\}$ |
| $Q_j$ | Queue length for operation $j$, $j \in \{1,2\}$ |
| $Q_j^{max}$ | Upper bound of queue length $j$, $j \in \{1,2\}$ |
| $C^h$ | Waiting cost rate of downstream operation 2<br>(Without losing generality, the waiting cost rate of upstream operation 1 is assumed to be 1.) |
| $\lambda$ | Job arrival rate at operation 1 |
| $\mu_i$ | Service rate for machines in group $i$ |
| $b_i$ | Failure rate for machines in group $i$, $b_i = \frac{1}{MTBF_i}$<br>defined by the mean time between failures (MTBF) of a machine |
| $r_i$ | Repair rate for machines in group $i$, $r_i = \frac{1}{MTTR_i}$<br>defined by the mean time to repair (MTTR) of a machine |
| $RL_i$ | Reliability of machines in group $i$<br>(portion of time when a machine is available in the long run) |
| $M_i$ | Number of available machines in group $i$, $M_i \in \{0,1,\dots,M_i^{max}\}$<br>$M_i$ changes in real time with machine health |

| | |
|---|---|
| $M_i^{max}$ | Number of machines in machine group $i$ |
| $a_t$ | Dispatching decision for flexible machines:<br>If $a_t = 0$, then assign a higher priority to upstream operation 1.<br>If $a_t = 1$, then assign a higher priority to downstream operation 2. |

Remark: Mean time between failures (MTBF), mean time to repair (MTTR), and reliability of machines are interchangeable on the basis of the following equation:

$$RL_i = \frac{MTBF_i}{MTBF_i + MTTR_i}, \forall i \in \{1,2,3\}.$$

In the current research, manufacturing decisions are made in real time after every manufacturing event. To perform real-time control after manufacturing events, Lippman (Lippman, 1975) recommends adopting *uniformization* to transform a continuous-time control problem into a discrete-time equivalent model that can be solved efficiently. The uniformization rate $\varphi$ used to derive the transition probabilities is defined as

$$\varphi = \lambda + \sum_{i=1}^{3} M_i^{max} \cdot (\mu_i + b_i + r_i).$$

In the discrete-time equivalent model, the time index $t$ represents a decision epoch that corresponds to each manufacturing event, which includes job arrival, service completion, machine failure/repair, and dummy events required by uniformization. The discrete-time equivalent MDP model is then solved through the commonly used value iteration algorithm with a time resolution that reaches the level of each manufacturing event. In accordance with Proposition 6.6.3 of Sennott (Sennott, 1998), the value iteration algorithm converges under our model assumptions to an optimal stationary control policy that assigns an optimal decision for every possible state at each decision epoch. As pointed out by Serfozo (Serfozo, 1978), the solution of the discrete-time model is equivalent to the optimal control policies of the continuous-time problem and can be adopted for the state-dependent real-time control of manufacturing systems.

Other key elements of the MDP model are defined as follows:

- Infinite planning horizon: $t \in \{1,2, \dots\}$
- The state at time *t* is denoted as $S_t = (Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t)$.
- Probability of job arrival: $P_1 = \lambda/\varphi$
- Probability of job completion of dedicated machines in group 1: $P_2 = (\min\{Q_1, M_1\} \cdot \mu_1)/\varphi$
- Probability of job completion of dedicated machines in group 2: $P_3 = (\min\{Q_2, M_2\} \cdot \mu_2)/\varphi$
- Probability of job completion of flexible machines in group 3 when $a_t = 0$: $P_4 = (\min\{Q_1, M_3\} \cdot \mu_3)/\varphi$
- Probability of job completion of flexible machines in group 3 when $a_t = 1$: $P_5 = (\min\{Q_2, M_3\} \cdot \mu_3)/\varphi$
- Probability of machine breakdown among group 1 dedicated machines: $P_6 = (M_1 \cdot b_1)/\varphi$
- Probability of machine breakdown among group 2 dedicated machines: $P_7 = (M_2 \cdot b_2)/\varphi$
- Probability of machine breakdown among group 3 flexible machines: $P_8 = (M_3 \cdot b_3)/\varphi$
- Probability of repair completion among group 1 dedicated machines: $P_9 = ((M_1^{max} - M_1) \cdot r_1)/\varphi$
- Probability of repair completion among group 2 dedicated machines: $P_{10} = ((M_2^{max} - M_2) \cdot r_2)/\varphi$

- Probability of repair completion among group 3 flexible machines: $P_{11} = ((M_3^{max} - M_3) \cdot r_3)/\varphi$
- Probability of the state remaining unchanged (dummy transition probability): $P_{dummy} = 1 - \sum_{i=1}^{11} P_i$

The cost function consists of the holding costs at each of the queue and can be denoted as $\mathrm{C}(S_t) = Q_1 + C^h \cdot Q_2$.

Then, the boundary condition, $V_{N+1}^*(Q_1, Q_2, M_1, M_2, M_3) = 0$, is defined. The backward optimality equation can be iteratively defined as

$$
\begin{aligned}
&V_t^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t) \\
&= C(S_t) + \min_{a_t \in \{0,1\}} \begin{Bmatrix} \{(1-a_t) \cdot (P_4 \cdot V_{t+1}^*(Q_1^t - 1, Q_2^t + 1, M_1^t, M_2^t, M_3^t) + P_5 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t)) \\ +a_t \cdot (P_4 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t) + P_5 \cdot V_{t+1}^*(Q_1^t, Q_2^t - 1, M_1^t, M_2^t, M_3^t)) \end{Bmatrix} \\
&\quad + P_1 \cdot V_{t+1}^*(Q_1^t + 1, Q_2^t, M_1^t, M_2^t, M_3^t) + P_2 \cdot V_{t+1}^*(Q_1^t - 1, Q_2^t + 1, M_1^t, M_2^t, M_3^t) \\
&\quad + P_3 \cdot V_{t+1}^*(Q_1^t, Q_2^t - 1, M_1^t, M_2^t, M_3^t) + P_6 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t - 1, M_2^t, M_3^t) \\
&\quad + P_7 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t - 1, M_3^t) + P_8 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t - 1) \\
&\quad + P_9 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t + 1, M_2^t, M_3^t) + P_{10} \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t + 1, M_3^t) \\
&\quad + P_{11} \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t + 1) + P_{dummy} \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t).
\end{aligned}
$$

The backward value iteration algorithm is then used to compute the optimal dynamic dispatching solution for each group of system parameters following Puterman (1994). The optimal action $a_t^*(S_t)$ of the current state can be formulated as

$$
\begin{aligned}
&a_t^*(S_t) = \\
&\mathrm{argmin}_{a_t \in \{0,1\}} \begin{Bmatrix} \{(1-a_t) \cdot (P_4 \cdot V_{t+1}^*(Q_1^t - 1, Q_2^t + 1, M_1^t, M_2^t, M_3^t) + P_5 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t)) \\ +a_t \cdot (P_4 \cdot V_{t+1}^*(Q_1^t, Q_2^t, M_1^t, M_2^t, M_3^t) + P_5 \cdot V_{t+1}^*(Q_1^t, Q_2^t - 1, M_1^t, M_2^t, M_3^t)) \end{Bmatrix}.
\end{aligned}
$$

Then, we present two numerical examples with the model parameters listed in Table 2 to illustrate the optimal dispatching policies generated from the MDP model.

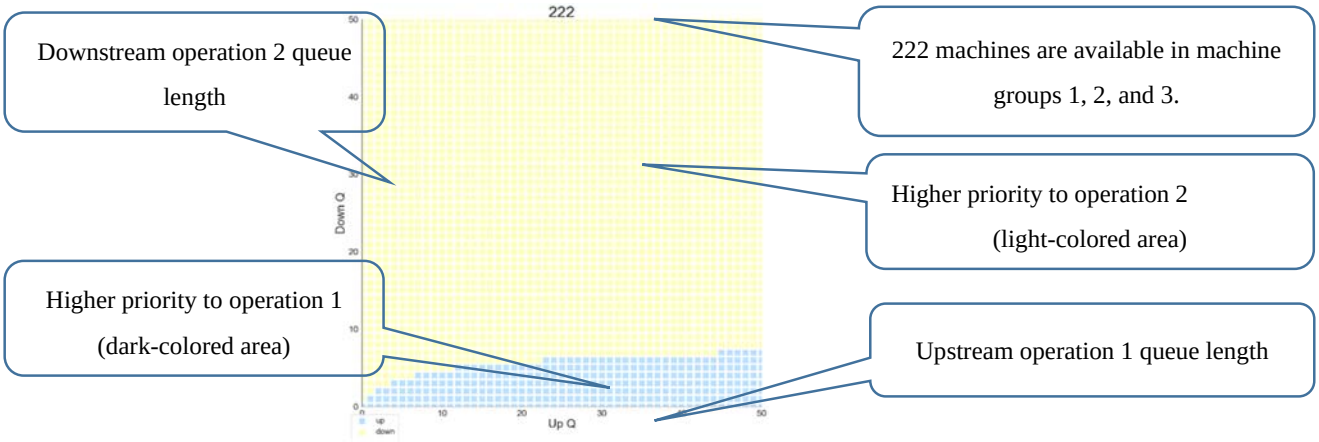
Table 2 System parameters used in the illustrative examples

| **System Parameter Set 1:** (corresponds to the optimal policy in Fig. 2) | | | | |
|---|---|---|---|---|
| Job Arrival Rate, $\lambda$ | | 10 | | |
| Upstream Holding Cost | | 1 | | |
| Downstream Holding Cost Ratio, $C^h$ | | 0.67 | | |
| | $MTBF_i$ | Service Rate ($\mu_i$) | Reliability ($RL_i$) | Number of Machines ($M_i^{max}$) |
| Group 1 Machines | 40 | 5 | 0.95 | 2 |
| Group 2 Machines | 40 | 5 | 0.95 | 2 |
| Group 3 Machines | 40 | 5 | 0.95 | 2 |
| | | | | |
| **System Parameter Set 2:** (corresponds to the optimal policy in Fig. 3) | | | | |
| Job Arrival Rate, $\lambda$ | | 8.58 | | |
| Upstream Holding Cost | | 1 | | |
| Downstream Holding Cost Ratio, $C^h$ | | 0.75 | | |
| | $MTBF_i$ | Service Rate ($\mu_i$) | Reliability ($RL_i$) | Number of Machines ($M_i^{max}$) |
| Group 1 Machines | 35 | 4 | 0.85 | 2 |
| Group 2 Machines | 35 | 4 | 0.85 | 2 |

| | | | | |
|---|---|---|---|---|
| Group 3 Machines | 35 | 4 | 0.85 | 2 |

To plot higher-dimension optimal policies into 2D graphs, we need to reduce the dimension by fixing the number of available machines in each machine group. After fixing the number of machines, we can then plot optimal decisions that correspond to changes in queue lengths. In both examples, the number of available machines varies from 0 to 2 because the maximum number of machines per machine group is two. Therefore, the visualization method introduced earlier provides $3 \times 3 \times 3 = 27$ decision graphs for both examples. In Figs. 2 and 3, the *X*-axis represents the queue length of upstream operation 1 and the *Y*-axis represents the queue length of downstream operation 2. We can then visualize the optimal policy under different numbers of available machines in Figs. 2 and 3. Each sub-decision graph shows the optimal decision under different combinations of available machines. In Figs. 2 and 3, the optimal action is to assign higher priority to operation 1 in darker areas. Notably, the number next to each sub-decision graph represents the number of available machines in each machine group, as shown in Fig. 2.

**Illustrative sub-decision graph (222): All machine groups have two available machines.**



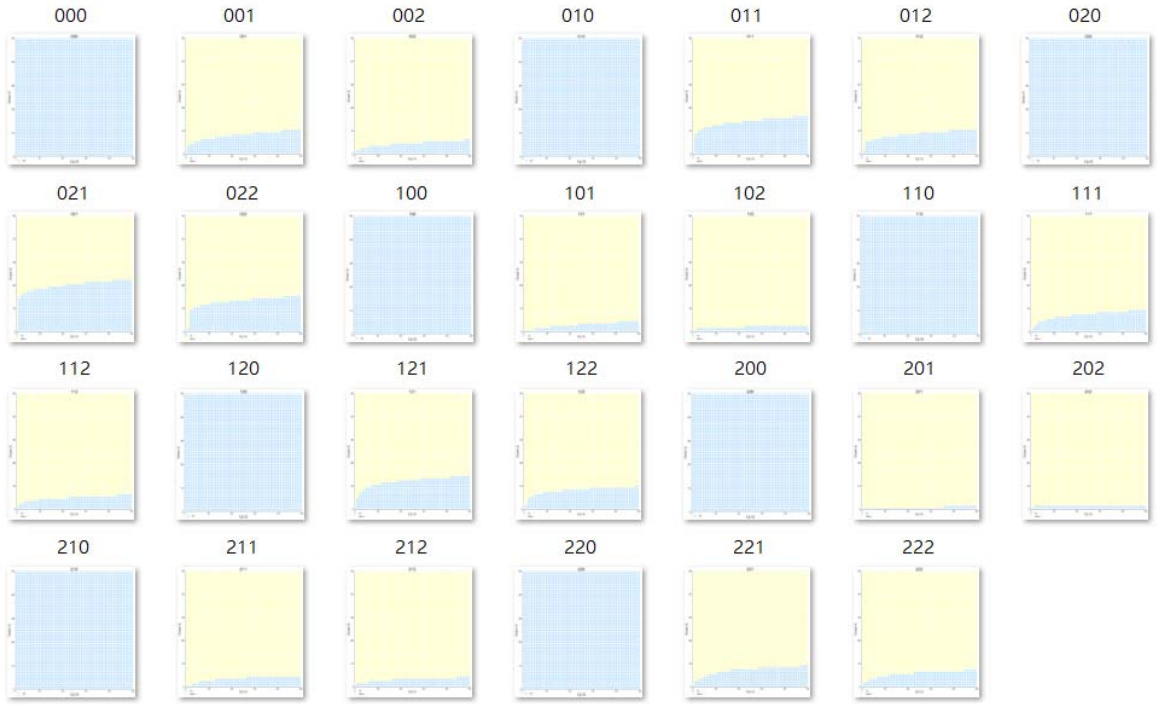



Fig. 2 Visualization of the optimal dispatching policy (system parameter set 1)

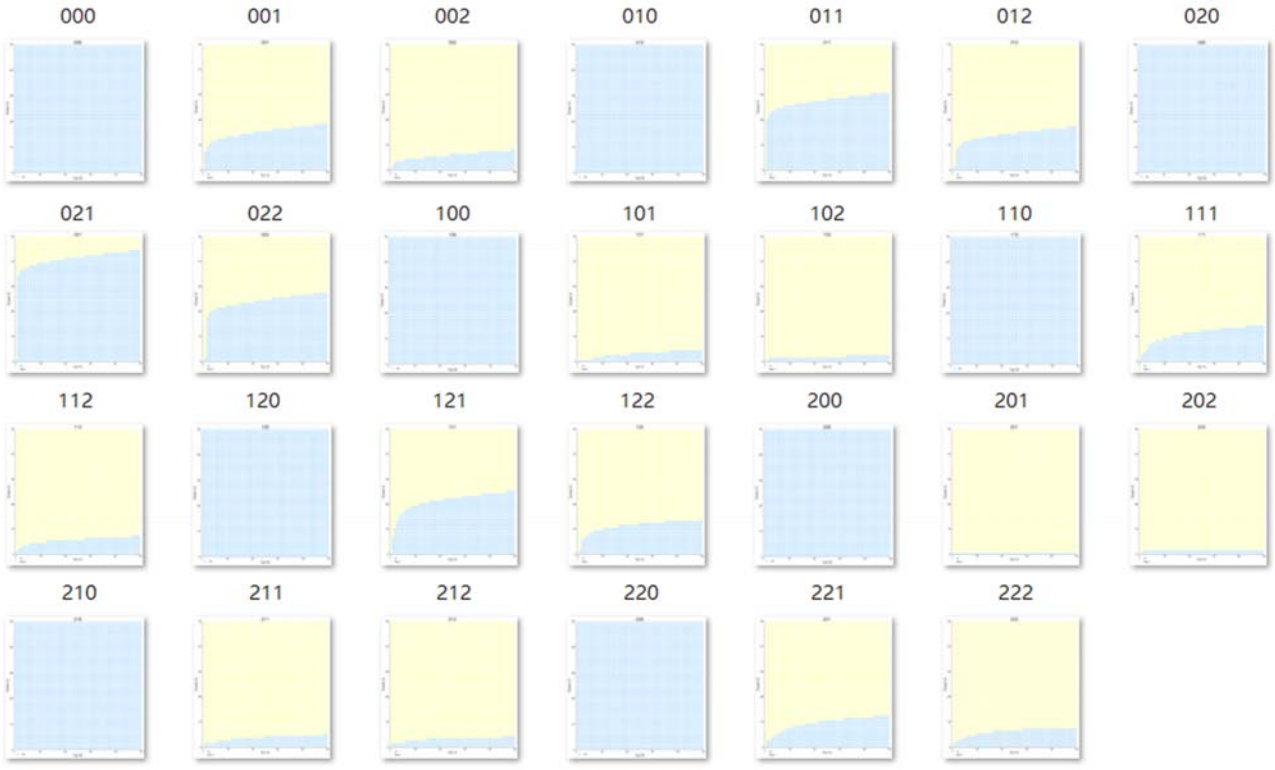
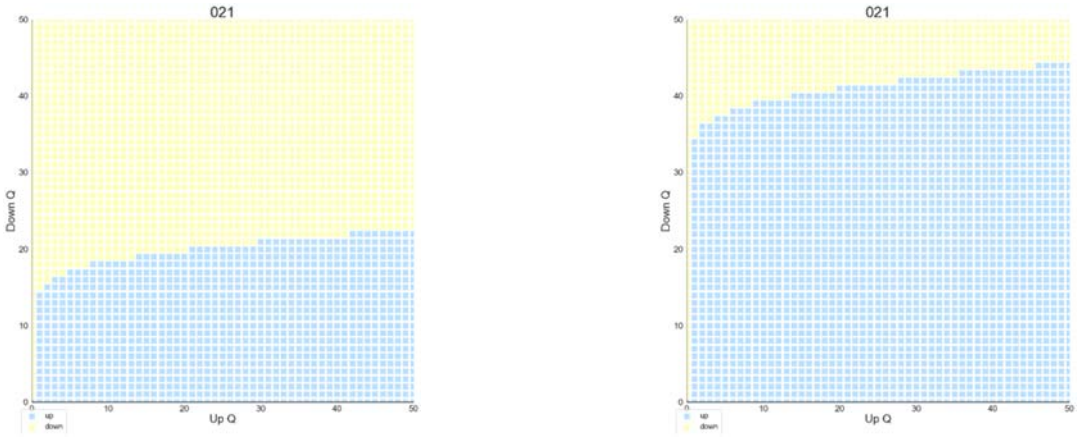



Fig. 3 Visualization of the optimal dispatching policy (system parameter set 2)

The barrier between dynamic programming models is a major challenge in dynamic production control. This barrier can be visualized by comparing the optimal policy graphs in Figs. 2 and3. The comparison of the two system parameter sets in Table 2 indicates that the numbers of machines in each machine group are the same, but other parameters are changed by 10%–20% in the second parameter set. Given that both systems

have six machines, the state space contains 140,454 states when the queue capacity is set to 50 for operations 1 and 2. Among the 140,454 states, 95% of the optimal decisions are the same. However, Fig. 4 shows that when no machine, 2 machines, and 1 machine are available in machine groups 1, 2, and 3, respectively, the dispatching decisions vary. Such difference is the so-called barrier between dynamic decision models. This barrier is observable after a mere 10% change of parameters.




Optimal decisions for parameter set 1 Optimal decisions for parameter set 2

Fig. 4 Barrier between MDP models: Optimal decisions vary with system parameters (0/2/1 machines are available in machine groups 1/2/3, respectively)

As shown in Figs. 2–4, minor changes in system parameters may lead to different optimal dispatching decisions. Given that the number of machines remains unchanged, the state space remains the same in Figs. 2 and 3. However, if we install another machine or change the qualification of machines, then the state space will differ, the barrier between models will be even more significant, and the computer code for solving the model may require rewriting.

Moreover, due to the curse of dimensionality of MDP, the required computational time for solving the dynamic dispatching problems increases exponentially with the number of machines as shown in Fig. 5. When configuration changes are frequent, both the curse of dimensionality and the barrier between dynamic programming models limit the use of dynamic control methods in the production environment.

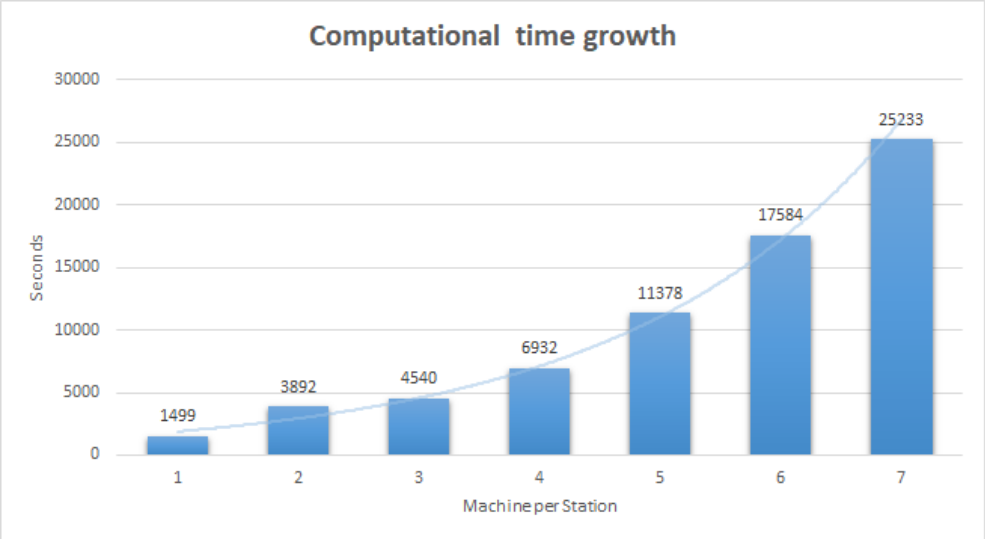


Fig. 5 Computational time increases with the number of machines in dynamic dispatching problems

## 3.2 DL approach for the dynamic dispatching problem

This research plans to develop a DL approach that can overcome the barrier between models and the curse of dimensionality shown in Figs. 2–5. We introduce the DL approach for the dynamic dispatching problem in this section. We begin by explaining why DNN can be used as an efficient predictor for optimal policies under different system configurations. Then, we describe how hyper-parameters, or parameters set prior to training, are selected and how training processes are implemented. This study uses various designs of optimal policy training samples to construct different DNNs. Discrete event simulation models are used to determine the performance difference between the dispatching policies predicted by the DNN and the optimal dispatching policies generated by dynamic programming. The standard K-fold cross-validation test (K-cv test) for machine learning is also used to test the prediction accuracy of the DL result.

The major difference between a classical artificial neural network (CANN) and DNN is uncomplicated. A commonly used CANN is a feedforward architecture with three fully connected layers: the input, hidden, and output layers. Backpropagation and stochastic gradient descent are used as the major algorithms to update weights between any linked neurons. Hornik, Stinchcombe, and White (1989) prove that the neural network is a universal approximator on the basis of this architecture. Compared with CANN, DNN introduces additional hidden layers and enables better fit to a complex nonlinear problem. However, given that we increase number of hidden layers, the performance of the model does not improve because of the vanishing gradient problem. Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014) propose an extremely effective method of training a neural network with many hidden layers without suffering from the vanishing gradient problem by randomly dropping out neurons while training. DL has achieved remarkable results in many fields following the aforementioned research. First, DL is widely used in image processing and classification. Krizhevsky, Sutskever, and Hinton (2012) construct a deep convolutional layer network with five convolutional layers to classify more than one million high-resolution images. Szegedy et al. (2015) propose a deep convolutional network with 22 layers for image classification. DNN is also used in optimization applications. Martens and Sutskever (2011) present an application of Hessian-free optimization using DL.

As shown in Fig. 6, we use DNN to learn from a large optimal policy dataset that contains low-dimensional optimal dispatching policies. If DNN can characterize the influence of model parameter changes on the optimal policy, then it can be used to predict optimal policies after a change occurs in system configuration. Moreover, the trained weights of DNN can be recorded, stored, and reused. Once a DNN for predicting an optimal policy is built, it can yield the best control action for each state depending on the real time system state. Hence, the barrier of being unable to reuse different dynamic programming models is overcome. When more machines are added, the state space of the MDP model increases and the curse of

dimensionality prevents the efficient resolution of the MDP model. An effective DNN policy predictor may rapidly generate predicted policies and eliminate the need to solve a time consuming large scale MDP model.

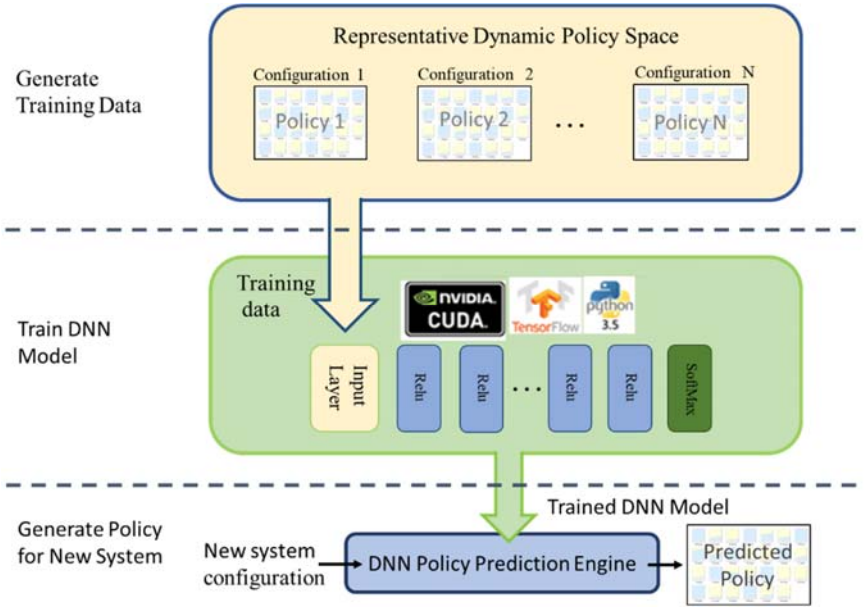



Fig. 6 DL framework for predicting the optimal control policies of different systems

In the proposed DNN model, the training data are the optimal dispatching policies under different system configurations. System configurations refer to the parameters or settings of a system, such as the MTBF/MTTR of machines, product mix, demand arrival processes, number of machines installed, and product–machine dedications. Optimal control policies assign a state-dependent dispatching decision for each and every possible system state in the state space, which is defined by the real-time WIP distribution and machine availability/reliability status, as shown in Figs. 2 and 3. When a new product or system configuration is introduced or observed, the optimal control problem must be solved again frequently to generate the corresponding new control policy.

In modern low-volume and high-mix (LVHM) production environments, training data should be readily available. Given that configuration changes, such as product mix changes or new product introduction, are frequent in LVHM systems, control problems under different configurations have been solved many times in the past. These historical control policies can serve as the required training data for the DNN model. By contrast, if historical control policies are not readily available, then training data can be generated by solving the MDP control problems under different synthetic configurations by increasing/decreasing the numbers of machines, MTBF, MTTR, or product mix. In this research, for example, our industry partner cannot provide all the historical control policies; thus, training data are generated by solving the MDP model under different configurations.

## 3.3 Representing Dynamic Policy Space: Training Dataset for DNN

To illustrate the potential of the DNN approach in dynamic production control, we build a knowledge space that allows DL to predict the best decisions for large systems (each workstation has four to five machines) by learning the optimal decisions for small-scale systems (each workstation has one to three machines). To

generate the training dataset, we use the Taguchi (1986) mixed orthogonal experimental design to generate a number of system parameter sets with one to three machines in each machine group. We then use the MDP model to solve the optimal dispatching problem for each set of system parameters and merge the results into a data file to construct the low-dimension optimal policy space. The low-dimension policy space can then be used by the DNN to learn the variations in system parameters by changing preconditions.

The sets of system factors and parameters with one to three machines in each machine group are provided in Table 3. The factors in Table 3 are used to generate the optimal policy space with 300 different systems.

**Dataset 1:** We generate 150 sets of small-scale system parameters with less than three machines in each machine group using the factors in Table 3 and a near-orthogonal experimental design.

**Dataset 2:** Another set of 150 near-orthogonal system parameters that emphasizes the number of machines in systems is generated because predicting optimal policies in large systems with more machines is a major objective of this study. Among the 150 additional systems, 3 sets of 50 systems are generated with exactly (1,1,1), (2,2,2), and (3,3,3) machines in machine groups 1, 2, and 3, respectively.

Table 3 Factors considered in the experiment

| **Factor** | **Levels** $\{low, relatively\ low, medium, relatively\ high, high\}$ |
|---|---|
| Arrival rate $\lambda$ (in terms of system utilization) | $\lambda \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$ *of the capacity* |
| MTBF of machine group 1 | $MTBF_1 \in \{20, 25, 30, 35, 40\}$ |
| MTBF of machine group 2 | $MTBF_2 \in \{20, 25, 30, 35, 40\}$ |
| MTBF of machine group 3 | $MTBF_3 \in \{20, 25, 30, 35, 40\}$ |
| Reliability of machine group 1 | $RL_1 \in \{0.85, 0.875, 0.9, 0.925, 0.95\}$ |
| Reliability of machine group 2 | $RL_2 \in \{0.85, 0.875, 0.9, 0.925, 0.95\}$ |
| Reliability of machine group 3 | $RL_3 \in \{0.85, 0.875, 0.9, 0.925, 0.95\}$ |
| Service rate of machine group 1 | $\mu_1 \in \{1, 2, 3, 4, 5\}$ |
| Service rate of machine group 2 | $\mu_2 \in \{1, 2, 3, 4, 5\}$ |
| Service rate of machine group 3 | $\mu_3 \in \{1, 2, 3, 4, 5\}$ |
| Holding cost ratio, $C^h$ | $C^h \in \{0.6667, 0.8, 1, 1.25, 1.5\}$ |
| $M_1^{\max}$, number of machines in machine group 1 | $M_1^{max} \in \{1, 2, 3\}$ |
| $M_2^{\max}$, number of machines in machine group 2 | $M_2^{max} \in \{1, 2, 3\}$ |
| $M_3^{\max}$, number of machines in machine group 3 | $M_3^{max} \in \{1, 2, 3\}$ |

We solve the dynamic dispatching problems for all the 300 systems using Microsoft Visual C# on a desktop PC with Intel Core i7 (3.5 GHz), 32 GB DDR3 memory to construct the optimal policy space as training data. We then explain how the DNN was implemented to characterize the optimal policy space.

## 3.4 DL approach for dynamic dispatching

The optimal policy space is a vector space that contains millions of states with its different systems and state features and the corresponding optimal action. To extract knowledge from the optimal policy space, we use TensorFlow as the policy approximator to predict the near-optimal policy of large-scale systems. The architecture of the DNN model is illustrated in Figs. 6 and 7.

The feature vector of the optimal policy space includes the following variables:

$(\lambda, \mu_1, \mu_2, \mu_3, MTBF_1, MTBF_2, MTBF_3, RL_1, RL_2, RL_3, C^h, M_1^{max}, M_2^{max}, M_3^{max}, Q_1, Q_2, M_1, M_2, M_3)^t$;

where the DNN approximates the optimal policy via nonlinear mapping $\tilde{a}_t$ as follows:

$(\lambda, \mu_1, \mu_2, \mu_3, MTBF_1, MTBF_2, MTBF_3, RL_1, RL_2, RL_3, C^h, M_1^{max}, M_2^{max}, M_3^{max}, Q_1, Q_2, M_1, M_2, M_3)^t$

$\rightarrow \tilde{a}_t(\lambda, \mu_1, \mu_2, \mu_3, MTBF_1, MTBF_2, MTBF_3, RL_1, RL_2, RL_3, C^h, M_1^{max}, M_2^{max}, M_3^{max}, Q_1, Q_2, M_1, M_2, M_3)$;

and $\theta^t$ denotes all the parameters and weights of the DNN. The training goal is to find $\theta^t$ that minimizes loss from the optimal dispatching decision $a_t^*$ to the nonlinear mapping $\tilde{a}_t$:

$a_t^*(\lambda, \mu_1, \mu_2, \mu_3, MTBF_1, MTBF_2, MTBF_3, RL_1, RL_2, RL_3, C^h, M_1^{max}, M_2^{max}, M_3^{max}, Q_1, Q_2, M_1, M_2, M_3)$

$\rightarrow \tilde{a}_t((\lambda, \mu_1, \mu_2, \mu_3, MTBF_1, MTBF_2, MTBF_3, RL_1, RL_2, RL_3, C^h, M_1^{max}, M_2^{max}, M_3^{max}, Q_1, Q_2, M_1, M_2, M_3|\theta^t)$.

For the DNN structure, we refer to the literature and use the following settings.

- Activation function: The activation function of hidden layer neurons is the rectified linear function (ReLU) provided in Nair and Hinton (2010). ReLU is a piecewise linear function that is commonly used in classification problems. ReLU can overcome the vanishing gradient problem in deep neural networks (DNNs) (Arora, Basu, Mianjy, & Mukherjee, 2016; Ramachandran, Zoph, & Le, 2017). The successive use of the nonlinear ReLU function in several hidden layers also allows the DNN to approximate any nonlinear function with an arbitrary precision (Leshno, Lin, Pinkus, & Schocken, 1993). In addition, we investigated the use of several activation functions, including the hyperbolic tangent and sigmoid functions and the combination of activation functions in different layers. Since the numerical results also suggest the superiority of ReLU in this specific application of DNN, ReLU is adopted for all the hidden layers in the proposed DNN model.
- Loss function: In this research, cross entropy is adopted as the loss function and the DNN model is trained to predict the optimal decision for every state in the state space. Given that each distinct dispatching decision can be represented by a different class, the prediction of the optimal decision can be considered classification problems. For the classification problems, the cross-entropy loss function is frequently used to measure the loss of classification models.
- Batch normalization: In densely connected DNNs, finding the conditional distribution of a hidden layer is difficult given a training data vector. This difficulty leads to an internal covariate shift and hampers

training performance. In our research, batch normalization is applied to accelerate training and reduce the impact of an internal covariate shift following Ioffe and Szegedy (Ioffe & Szegedy, 2015).

- Learning rate and random dropout: To prevent overfitting, a random dropout mechanism is used with a rate of 0.25 and the batch size of the input vectors is 125 based on Srivastava et al. (2014). We set an initial learning rate of 0.25 in DNN training and apply callbacks to save the neural network model before it becomes overfitted. Callback is implemented by reducing the learning rates by 30% as shown in Fig. 7.

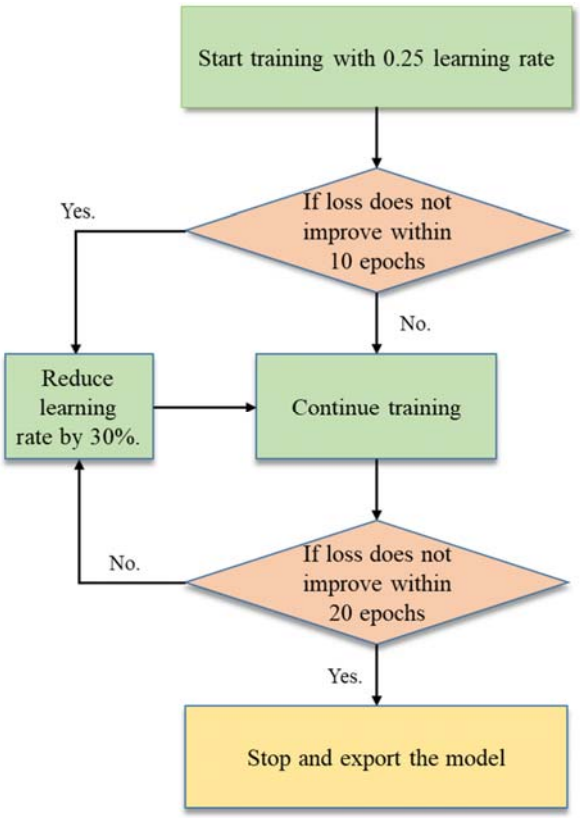
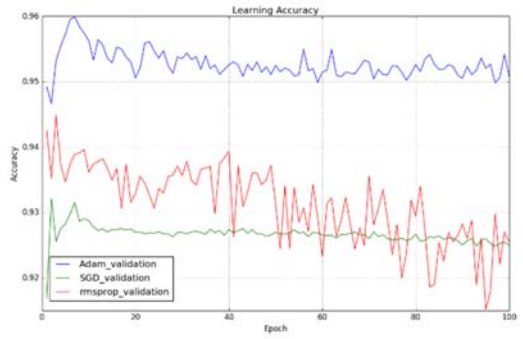



Fig. 7 Reducing learning rate to prevent overfitting

- Optimizers, hidden layers, and number of neurons: Using the first datatset of 150 near-orthogonal systems, we test three commonly adopted DNN optimizers, namely, stochastic gradient descent (SGD), adaptive moment estimation (ADAM) from Kingma and Ba [32], and RMSprop from Tieleman and Hinton [33], to optimize the categorical cross-entropy loss function. We measure the performance of DL by using a K-cv test with 30% of the training data for validation. The learning result using different optimizers is shown in Figs. 8.a and 8.b, which indicate that ADAM outperforms the other optimizers. Therefore, ADAM is selected as our optimizer for DL.

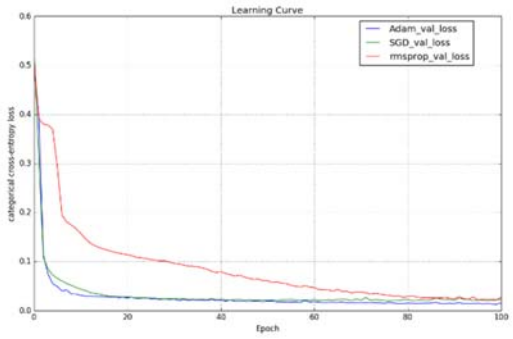
  For the layers and neurons, Heaton [32] states that the optimal hidden layer size is at most the size of the input layers and at least the size of the output layers. By using a K-cv test with 30% of the training data for validation, Fig. 8.c shows how different structures of DNN influence K-cv accuracy. From the results in Fig. 8.c, a DNN with 8 hidden layers with each layer containing 90 neurons is found better performing. This DNN achieves 95.12% accuracy, and its structure is used for our remaining computational studies.

In the succeeding section, we use discrete event simulation to test the actual performance of the control policy and compare dynamic programming with DL.



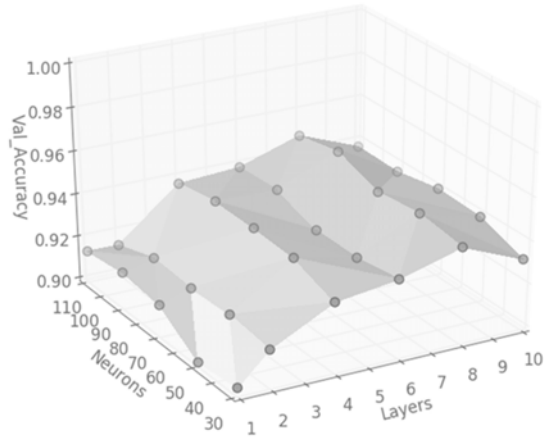

a. K-cv accuracy

(Dataset 1: 150 near-orthogonal systems)



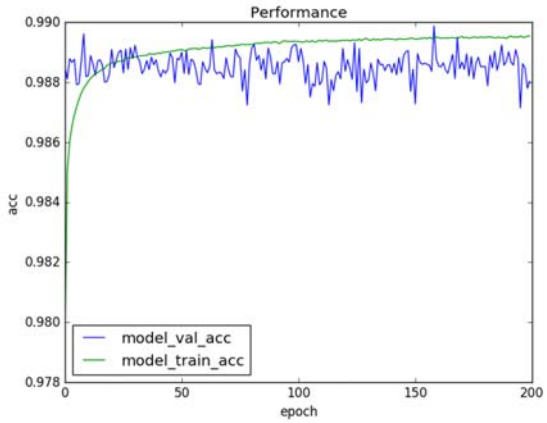

b. Cross-entropy loss under ADAM, SGD, and RMSProp

(Dataset 1: 150 near-orthogonal systems)




c. Accuracy under different DNN structures

(Dataset 1: 150 near-orthogonal systems)




d. Accuracy using a combined dataset of 300 systems

(Datasets 1 and 2: 150 near-orthogonal systems and additional 150 systems)

Fig. 8 Training performance of different optimizers, DNN configurations, and datasets

Using the hyperparameters selected earlier, we train the same DNN with 8 hidden layers using the complete dataset of 300 systems with Datasets 1 and 2. Learning performance is shown in Fig. 8.d. With GPU acceleration via nVidia GTX 1060, the computational cost of 1 epoch is 360 s on average when using the complete dataset and the overall training time is less than 20 h for 200 epochs. Notably, the performance of the trained DNN model improves only slightly after the first 50 epochs, as shown in Fig. 8.d. This finding suggests that training time can be reduced further to 5 h if it is a consideration.

Compared with the learning performance with only 150 sets of near-orthogonal system parameters, the K-cv test score improves to 99% in Fig. 8.d from the maximum of 95.12% in Fig. 8.c. Given that the same DNN is used, such improvement is achieved by including training Dataset 2. This result suggests that training dataset selection and the structure of a neural network are important in knowledge extraction through DNN.

# 4. Evaluation of Model Accuracy and Performance

In this section, we analyze the similarity between the near-optimal policy generated by the DNN and the real optimal policy determined by dynamic programming. Thereafter, we construct a discrete event simulator to further observe the difference between DL and dynamic programming.

We randomly generated 15 test systems using the machine and production parameters listed in Table 3. Table 4 shows the system parameters used in the 15 test systems. Although the training of the DNN considers only up to three machines in each machine group (i.e., $M_i^{max} \leq 3$), the number of available machines can be increased to five in each machine group in the 15 test systems. The number of machines $M_i^{max}$ in systems 1–5 uses the same range as that in the training dataset (i.e., $M_i^{max}$ ranging from one to three in each machine group). Test systems 1–5 are named the interpolation test group because the number of machines is interpolated within the range of the training dataset. Systems 6–10 are used for the extrapolation test because the number of machines $M_i^{max}$ is randomly generated between four and five although the training data only allow up to three machines. Test systems 11–15 are named the complete testing set because the number of machines in each machine group is randomly selected from one to five. The interpolation and extrapolation of the number of machines are both possible in systems 11–15. In summary, the interpolation test group assesses the performance of the DNN model when the system parameters are interpolated within the range of the training dataset, whereas the extrapolation test group validates the DNN model performance when the system parameters are extrapolated beyond the range of the training dataset.

Table 4 Model parameters of the 15 test systems

| | No. | λ | $u_1$ | $MTBF_1$ | $RL_1$ | $u_2$ | $MTBF_2$ | $RL_2$ | $u_3$ | $MTBF_3$ | $RL_3$ | Cost2 | $Mmax_1$ | $Mmax_2$ | $Mmax_3$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Interpolation | 1 | 0.84 | 1 | 29 | 0.95 | 4 | 40 | 0.88 | 1 | 20 | 0.95 | 1.13 | 3 | 1 | 1 |
| | 2 | 0.76 | 3 | 24 | 0.86 | 5 | 32 | 0.94 | 2 | 36 | 0.92 | 1.22 | 2 | 1 | 2 |
| | 3 | 0.79 | 1 | 32 | 0.95 | 1 | 21 | 0.9 | 1 | 24 | 0.92 | 1.49 | 2 | 1 | 3 |
| | 4 | 0.87 | 5 | 22 | 0.89 | 2 | 22 | 0.87 | 1 | 29 | 0.86 | 1.35 | 2 | 2 | 1 |
| | 5 | 0.76 | 3 | 32 | 0.87 | 1 | 29 | 0.91 | 3 | 26 | 0.9 | 0.86 | 2 | 3 | 2 |
| Extrapolation | 6 | 0.76 | 5 | 39 | 0.88 | 1 | 23 | 0.94 | 3 | 20 | 0.89 | 1.03 | 5 | 4 | 4 |
| | 7 | 0.71 | 5 | 24 | 0.87 | 4 | 30 | 0.85 | 3 | 23 | 0.93 | 1.03 | 4 | 5 | 5 |
| | 8 | 0.82 | 4 | 28 | 0.85 | 4 | 38 | 0.85 | 5 | 26 | 0.92 | 1.31 | 4 | 5 | 5 |
| | 9 | 0.78 | 2 | 32 | 0.87 | 3 | 35 | 0.87 | 3 | 24 | 0.91 | 0.91 | 4 | 4 | 5 |
| | 10 | 0.9 | 4 | 28 | 0.91 | 2 | 26 | 0.9 | 4 | 29 | 0.86 | 0.99 | 4 | 5 | 4 |
| Complete Set | 11 | 0.75 | 4 | 35 | 0.92 | 2 | 23 | 0.95 | 4 | 23 | 0.92 | 0.9 | 3 | 4 | 3 |
| | 12 | 0.78 | 3 | 39 | 0.86 | 1 | 30 | 0.91 | 2 | 26 | 0.83 | 0.83 | 4 | 1 | 5 |
| | 13 | 0.81 | 1 | 39 | 0.93 | 3 | 24 | 0.86 | 4 | 20 | 0.94 | 0.83 | 3 | 2 | 5 |
| | 14 | 0.82 | 5 | 23 | 0.91 | 4 | 32 | 0.89 | 4 | 23 | 0.88 | 0.71 | 4 | 1 | 4 |
| | 15 | 0.77 | 2 | 20 | 0.9 | 4 | 38 | 0.94 | 1 | 25 | 0.89 | 0.92 | 4 | 5 | 3 |

## 4.1 Accuracy of the DNN approach

The accuracy of the DNN approach is measured by the similarity between the policies generated by the DNN and the optimal policies generated by MDP. For all 15 test systems, we use the DNN shown in Fig. 8.d, which is trained by a complete dataset of 300 systems with 8 hidden layers and 90 neurons in each layer, to generate predicted optimal control policies. As shown in Table 5, the DNN takes 10.23 s to generate a predicted policy, which is considerably faster than the MDP model, which requires more than 10,000 s to solve large problems (e.g., test systems 6–10). As illustrated in Fig. 5, the required computational time is more than 1,000s,

even for small systems with only one machine per machine group, which is still 100 times the DNN output time. Thus, in terms of computational efficiency, the DNN provides dynamic control policies nearly instantly in our numerical study and is 100–1,000 times faster than MDP models in generating dynamic control policies.

In addition to computational efficiency, the DNN approach performs well in solution quality. Among the 15 test systems, the overall similarity between the DNN policy and the optimal policy is 98.36%. In the worst case scenario, accuracy is 96.16% in test system 15.

Table 5 Computation time and accuracy of DNN policy prediction

| System | Similarity to Optimal (Accuracy, %) | DNN output time (seconds) | Correct Prediction # of states | Incorrect Prediction # of states |
|---|---|---|---|---|
| **1** | 97.46% | 2.77 | 40563 | 1053 |
| **2** | 96.82% | 2.99 | 45328 | 1490 |
| **3** | 99.82% | 3.59 | 62316 | 108 |
| **4** | 99.44% | 2.93 | 46557 | 261 |
| **5** | 99.34% | 4.81 | 93014 | 622 |
| **6** | 98.29% | 16.91 | 383466 | 6684 |
| **7** | 98.24% | 20.08 | 459959 | 8221 |
| **8** | 97.01% | 19.94 | 454206 | 13974 |
| **9** | 97.47% | 17.04 | 380298 | 9852 |
| **10** | 97.97% | 16.77 | 382236 | 7914 |
| **11** | 98.51% | 9.47 | 204969 | 3111 |
| **12** | 99.85% | 7.41 | 155833 | 227 |
| **13** | 99.41% | 8.66 | 186181 | 1091 |
| **14** | 99.59% | 6.31 | 129521 | 529 |
| **15** | 96.16% | 13.74 | 300150 | 11970 |
| ***Average*** | ***98.36%*** | ***10.23*** | ***Total 3234137*** | ***Total 79537*** |

Notably, the difference between DNN and optimal policies is mostly located at the boundary of dark and light areas. Using test system 1 (with an average accuracy of 97.46%) as an example, we plot the comparison of the DNN policy and the optimal policy in Fig. 9, where the red areas indicate the difference between the two policies. Wu et al. (C.-H. Wu et al., 2006) show that the optimal dispatching decisions switch from one action to another after the queue length reaches a certain threshold value because the cost difference between two decisions is monotone to queue length. Thus, the boundary areas indicate the region where the cost difference between the two decisions is small and predicting the optimal action is relatively difficult for the DNN. Given that prediction errors mostly appear at the boundary states where the cost difference is small, we strongly believe that the cost difference between the DNN policy and the optimal policy is minimal. Therefore, we construct a discrete event simulation model to compare the costs of the two policies.

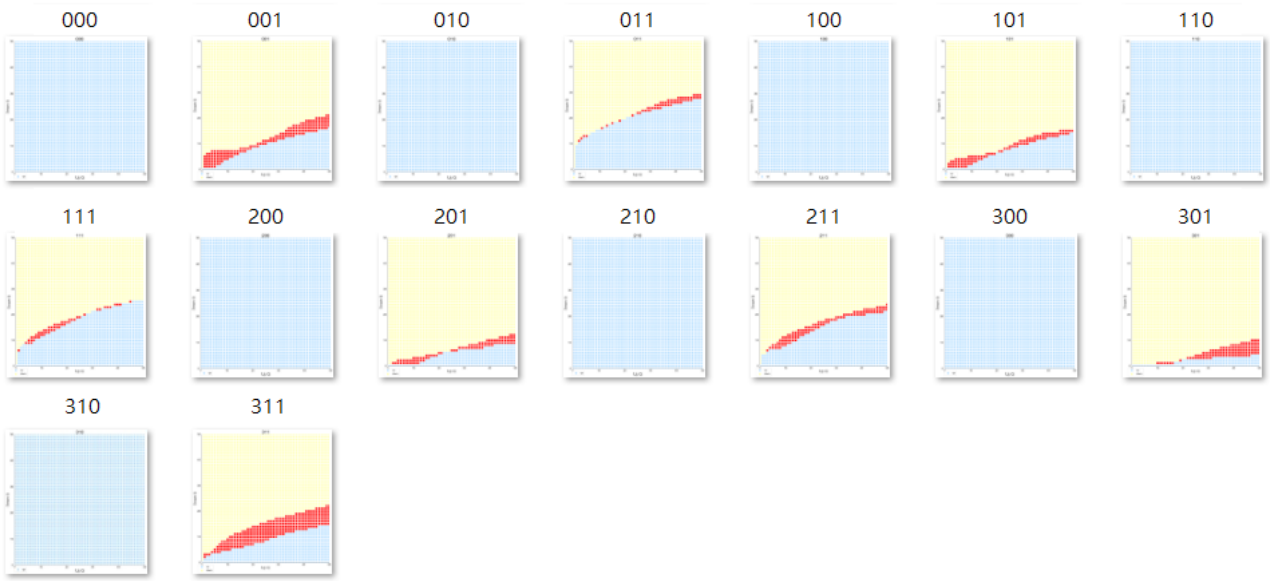



Fig. 9 Visualized erroneous decision of the DNN model: Test system 1 in Table 5

*(Please refer to Fig. 2 for the details on the legends and captions of subgraphs.)*

## 4.2 Performance evaluation using discrete event simulation

In addition to model accuracy, the performance of the DNN policy is compared with the optimal policy through simulation. We use the eM-plant to implement our discrete event simulator to further verify the performance of near-optimal policies. We apply the policies produced from dynamic programming and DNN to a plant simulation model. The simulation time is set to 90 days with a 7-day warm-up period. Each simulation is repeated 50 times, and the average results are listed in Table 6. The weighted cycle time (i.e., inventory holding costs) is the key performance index.

The results in Table 6 indicate that the holding cost (weighted cycle time) difference between DL and dynamic programming is 1.33% on average in the 15 systems. Overall, the difference in weighted cycle time is lower than 1% in 8 of the 15 systems, and the difference is less than 5% in all the systems. Given that the solution gap is small in all the systems, we can conclude that the DNN model is effective in generating dynamic control policies for large systems that may suffer from the curse of dimensionality of MDP. Moreover, we conclude that the DNN approach can also break the barrier among MDP models shown in Fig. 5 because new policies can be generated instantly after each system configuration is changed. Consequently, the frequent requirement to solve MDP models can be eliminated.

Table 6 Simulation results: Difference in performance between the DNN policy and the optimal DP policy (The 95% confidence interval of the holding costs/weighted cycle time is included)

| Group | | Average Waiting Time | | Average Process Time | | | Holding Costs ± *95% C.I.* |
|---|---|---|---|---|---|---|---|
| | | Up-Queue | Down-Queu | Up-stream | down-Stream | Flexible | (Weighted Cycle Time) |
| 1 | DP | 50286.9 | 13285.1 | 3062.7 | 955.1 | 933.4 | 1538557 ± *9458* |
| | NN | 51183.7 | 13578.3 | 3104.7 | 981.8 | 899.4 | 1546780 ± *8660* |
| | Difference | 1.75% | 2.16% | 1.35% | 2.72% | -3.78% | 0.53% |
| 2 | DP | 951.4 | 973.2 | 708 | 362.6 | 258.4 | 101691 ± *2913* |
| | NN | 899.2 | 1035.9 | 704.1 | 365.2 | 256.1 | 102966 ± *2006* |
| | Difference | -5.81% | 6.05% | -0.55% | 0.71% | -0.90% | 1.24% |
| 3 | DP | 8601.6 | 6321.5 | 2854.6 | 1549.4 | 2849.9 | 247341 ± *11301* |
| | NN | 8338.4 | 6394.2 | 2825.9 | 1564.2 | 2789.2 | 249459 ± *10256* |
| | Difference | -3.16% | 1.14% | -1.02% | 0.95% | -2.18% | 0.85% |
| 4 | DP | 588.3 | 3824.5 | 719.9 | 1266.7 | 482.3 | 186293 ± *9485* |
| | NN | 612.2 | 3777.6 | 722 | 1261.7 | 479.9 | 187005 ± *6690* |
| | Difference | 3.90% | -1.24% | 0.29% | -0.40% | -0.50% | 0.38% |
| 5 | DP | 28529.3 | 4488 | 1146.5 | 1710.2 | 863.5 | 1346038 ± *1698* |
| | NN | 28507.6 | 4939.9 | 1145.5 | 1711.7 | 864.2 | 1362536 ± *1937* |
| | Difference | -0.08% | 9.15% | -0.09% | 0.09% | 0.08% | 1.21% |
| 6 | DP | 65.6 | 1735.8 | 784.9 | 1283.5 | 939.8 | 131571 ± *2389* |
| | NN | 63.9 | 1827.9 | 735.4 | 1292.6 | 941.6 | 138127 ± *2666* |
| | Difference | -2.66% | 5.04% | -6.73% | 0.70% | 0.19% | 4.75% |
| 7 | DP | 163.7 | 220.7 | 523.9 | 813.4 | 661.8 | 44013 ± *903* |
| | NN | 133.7 | 260.9 | 495.1 | 845 | 665.4 | 45119 ± *1259* |
| | Difference | -22.44% | 15.41% | -5.82% | 3.74% | 0.54% | 2.45% |
| 8 | DP | 9271.3 | 1158.3 | 746.8 | 921.1 | 762.3 | 1378230 ± *738* |
| | NN | 9261.9 | 1259.8 | 746 | 922.8 | 762.6 | 1395556 ± *699* |
| | Difference | -0.10% | 8.06% | -0.11% | 0.18% | 0.04% | 1.24% |
| 9 | DP | 18.7 | 95.4 | 1056.7 | 1221.8 | 724.5 | 2707 ± *104* |
| | NN | 11.1 | 131.4 | 1009.3 | 1268.2 | 723.5 | 2824 ± *132* |
| | Difference | -68.47% | 27.40% | -4.70% | 3.66% | -0.14% | 4.16% |
| 10 | DP | 10782.6 | 1683.1 | 885.5 | 1106.1 | 660.9 | 1345999 ± *3230* |
| | NN | 10795.8 | 1676.8 | 885.1 | 1105.9 | 662.1 | 1348163 ± *3937* |
| | Difference | 0.12% | -0.38% | -0.05% | -0.02% | 0.18% | 0.16% |
| 11 | DP | 740.8 | 1318.8 | 804.5 | 1351.3 | 868.5 | 114454 ± *3574* |
| | NN | 659.5 | 1466.4 | 789.72 | 1375.8 | 869.5 | 117448 ± *3262* |
| | Difference | -12.33% | 10.07% | -1.87% | 1.78% | 0.12% | 2.55% |
| 12 | DP | 1525.9 | 6649.5 | 1355.3 | 468.9 | 1747.1 | 349126 ± *10813* |
| | NN | 1512.6 | 6471 | 1351.4 | 458.4 | 1794.8 | 349530 ± *11267* |
| | Difference | -0.88% | -2.76% | -0.29% | -2.29% | 2.66% | 0.12% |
| 13 | DP | 17791.4 | 1539.4 | 1071.7 | 666.5 | 838.7 | 1269054 ± *507* |
| | NN | 17858.8 | 1507.7 | 1076 | 668.3 | 842.3 | 1267095 ± *427* |
| | Difference | 0.38% | -2.10% | 0.40% | 0.27% | 0.43% | -0.15% |
| 14 | DP | 14.3 | 1901.1 | 574.5 | 535.8 | 644.2 | 37299 ± *2234* |
| | NN | 20.3 | 1790.6 | 604.7 | 544.1 | 661.2 | 37384 ± *2218* |
| | Difference | 29.56% | -6.17% | 4.99% | 1.53% | 2.57% | 0.23% |
| 15 | DP | 792.8 | 26.7 | 1411.2 | 946.4 | 1226.6 | 40405 ± *1594* |
| | NN | 784.9 | 25.9 | 1406.5 | 945.4 | 1223.8 | 40514 ± *1423* |
| | Difference | -1.01% | -3.09% | -0.33% | -0.11% | -0.23% | 0.27% |

# 5. Conclusion and Future Research

This research combines DNN and MDP for the fast generation of near-optimal policies in large dynamic dispatching problems that are traditionally difficult to solve. MDP is used to generate a set of optimal training policies in small systems. To construct a meaningful set of training systems, an orthogonal experimental design technique is adopted. In contrast with other RL or Deep Q-learning research that learns the characteristics of value functions, the proposed DNN learns directly from the representative policy space and serves as an effective predictor of near-optimal control policy for larger systems.

To verify solution quality against optimal policies, the K-cv test is used to examine accuracy and discrete event simulation is adopted to compare costs. Compared with the optimal control policies, the overall accuracy of the DNN control policies is 98.36% on average. Moreover, the simulation results suggest a cost difference lower than 5% between the DNN and optimal policies in all test systems and less than 1.33% on average. The findings suggest that the policy produced by the DNN is similar to the optimal policy. The AI approach generates policies nearly instantly and is considerably more efficient than the traditional MDP approach.

This research demonstrates the potential of machine learning in dynamic production control; however, it is merely a first step toward the intelligent control of systems. The proposed DNN model learns from existing optimal control policies, and can be quickly extended to systems with more than two product types when the training data is available. The optimal control for systems with more than two products is itself a research challenge and most existing control methods use simple control heuristics for such systems. As a result, the optimal control policies that serve as the training data of the DNN cannot be efficiently generated by existing methods yet. Thus, although our preliminary numerical results have shown the potential of the proposed DNN approach in systems with more products, we limit our numerical analysis to two-product examples. In the future, we plan to implement our methods to consider more product types by developing an efficient decomposition algorithm for such systems.

Another limitation of the proposed DNN approach is the 2% gap in solution quality caused by the interpolation or extrapolation of system parameters. Further reducing the gap by fine-tuning the DNN model is difficult, and the possibility of using different neural network structures can be explored to improve model accuracy. Thus, we plan to combine the off-line DNN approach with online RL. After a quick adoption of off-line policies generated by the DNN model along with configuration changes, an online learning approach will be developed in the future to improve solution quality and eliminate the solution gap over time.

Moreover, in this study, we investigated a two-station production system with a fixed routing. Some of the machines are flexible and can be assigned to each of the two workstations. The proposed DNN is trained to effectively predict the optimal dynamic dispatching decisions of flexible machines. After adding new machines or introducing new products, the DNN generates predicted policies that are within 2% of the optimal range, thereby eliminating the need to solve large MDP problems. However, the DNN model needs to undergo the model training processes again when there are changes in the production system layout, such as additional workstations. DNN retraining can help in capturing new optimal control characteristics under the new

production system layout. We will develop transfer learning methods in future studies to accelerate DNN retraining.

# References


Ahn, H. S., Duenyas, I., & Zhang, R. Q. (1999). Optimal stochastic scheduling of a two-stage tandem queue with parallel servers. *Advances in Applied Probability, 31*(4), 1095-1117. doi:Doi 10.1017/S0001867800009642

Archibald, T. W. (2007). Modelling replenishment and transshipment decisions in periodic review multilocation inventory systems. *Journal of the Operational Research Society, 58*(7), 948-956.

Arora, R., Basu, A., Mianjy, P., & Mukherjee, A. (2016). Understanding Deep Neural Networks with Rectified Linear Units. *ArXiv: 1611.01491*.

Aydin, M. E., & Oztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems, 33*(2-3), 169-178. doi:Doi 10.1016/S0921-8890(00)00087-7

Chang, S. H., & Gershwin, S. B. (2010). Modeling and analysis of two unreliable batch machines with a finite buffer in between. *Iie Transactions, 42*(6), 405-421. doi:10.1080/07408170903228934

Choi, J. Y., & Kim, S. B. (2012). Computationally efficient neuro-dynamic programming approximation method for the capacitated re-entrant line scheduling problem. *International Journal of Production Research, 50*(8), 2353-2362. doi:10.1080/00207543.2011.578596

Gershwin, S. B. (1987). An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking. *Operations Research, 35*(2), 291-305. doi:DOI 10.1287/opre.35.2.291

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural networks, 2*(5), 359-366. doi:Doi 10.1016/0893-6080(89)90020-8

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv: 1611.01491*, 11.

Kirkizlar, H. E. (2008). *Performance improvements through flexible workforce.* Georgia Institute of Technology,

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks.* Paper presented at the Advances in neural information processing systems.

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function. *Neural networks, 6*(6), 861-867. doi:Doi 10.1016/S0893-6080(05)80131-5

Lippman, S. A. (1975). Apply a New Device in the Optimization of Exponential Queueing Systems. *Operations Research, 23*(4), 24.

Marbach, P., Mihatsch, O., & Tsitsiklis, J. N. (2000). Call admission control and routing in integrated services networks using neuro-dynamic programming. *IEEE Journal on selected areas in communications, 18*(2), 197-208. doi:Doi 10.1109/49.824797

Marbach, P., & Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control, 46*(2), 191-209. doi:Doi 10.1109/9.905687

Martens, J., & Sutskever, I. (2011). *Learning recurrent neural networks with hessian-free optimization.* Paper presented at the Proceedings of the 28th International Conference on Machine Learning (ICML-11).

Meyn, S. P. (2005). Workload models for stochastic networks: Value functions and performance evaluation. *IEEE Transactions on Automatic Control, 50*(8), 1106-1122. doi:10.1109/Tac.2005.852564

Mishra, N., Singh, A., Kumari, S., Govindan, K., & Ali, S. I. (2016). Cloud-based multi-agent architecture for effective planning and scheduling of distributed manufacturing. *International Journal of Production Research, 54*(23), 7115-7128. doi:10.1080/00207543.2016.1165359

Mouelhi-Chibani, W., & Pierreval, H. (2010). Training a neural network to select dispatching rules in real time. *Computers & Industrial Engineering, 58*(2), 249-256. doi:10.1016/j.cie.2009.03.008

Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted boltzmann machines.* Paper presented at the Proceedings of the 27th international conference on machine learning (ICML-10).

Nattaf, M., Dauzere-Peres, S., Yugma, C., & Wu, C. H. (2019). Parallel machine scheduling with time constraints on machine qualifications. *Computers & Operations Research, 107*, 61-76. doi:10.1016/j.cor.2019.03.004

Olafsson, S., & Li, X. A. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics, 128*(1), 118-126. doi:10.1016/j.ijpe.2010.06.004

Priore, P., Parreno, J., Pino, R., Gomez, A., & Puente, J. (2010). Learning-Based Scheduling of Flexible Manufacturing Systems Using Support Vector Machines. *Applied Artificial Intelligence, 24*(3), 194-209. doi:10.1080/08839510903549606

Puterman, M. L. (1994). Markov Decision Processes. J. In: Wiley and Sons.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for Activation Functions. *ArXiv: 1710.05941*.

Sennott, L. I. (1998). *Stochastic Dynamic Programming and the Control of Queueing Systems*: John Wiley & Sons.

Serfozo, R. (1978). An equivalence between continuous and discrete time Markov decision processes. *Operations Research, 27*(3), 5.

Shi, L., Guo, G., & Song, X. (2019). Multi-agent based dynamic scheduling optimisation of the sustainable hybrid flow shop in a ubiquitous environment. *International Journal of Production Research*, 1-22. doi:10.1080/00207543.2019.1699671

Shiue, Y. R. (2009). Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. *International Journal of Production Research, 47*(13), 3669-3690. doi:10.1080/00207540701846236

Simester, D. I., Sun, P., & Tsitsiklis, J. N. (2006). Dynamic catalog mailing policies. *Management Science, 52*(5), 683-696. doi:10.1287/mnsc.1050.0504

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research, 15*(1), 1929-1958.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). *Going deeper with convolutions.* Paper presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

Taguchi, G. (1986). *Introduction to quality engineering: designing quality into products and processes*.

Tsitsiklis, J. N., & Van Roy, B. (1999). Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control, 44*(10), 1840-1851. doi:Doi 10.1109/9.793723

Wu, C.-H., Lewis, M. E., & Veatch, M. (2006). Dynamic allocation of reconfigurable resources ina two-stage Tandem queueing system with reliability considerations. *IEEE Transactions on Automatic Control, 51*(2), 309-314.

Wu, C. H., Down, D. G., & Lewis, M. E. (2008). Heuristics for allocation of reconfigurable resources in a serial line with reliability considerations. *Iie Transactions, 40*(6), 595-611. doi:10.1080/07408170701744819

Wu, C. H., Lin, J. T., & Chien, W. C. (2010). Dynamic production control in a serial line with process queue time constraint. *International Journal of Production Research, 48*(13), 3823-3843. doi:10.1080/00207540902922836

Xia, B., Wang, C., Gao, Y., Peng, Y., & Liu, L. (2019). A new approach to the analysis of homogeneous transfer lines with unreliable buffers subject to time-dependent failure. *International Journal of Production Research*, 1-17. doi:10.1080/00207543.2019.1685700

Zhang, J., & Wang, X. (2016). Multi-agent-based hierarchical collaborative scheduling in re-entrant manufacturing systems. *International Journal of Production Research, 54*(23), 7043-7059. doi:10.1080/00207543.2016.1194535

Zhou, F.-Y., Wu, C.-H., & Yu, C.-J. (2017). Dynamic Dispatching for Re-entrant Production Lines - a Deep Learning Approach. *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 1026-1031.