Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 51 (2020) 1462–1468

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021)
15-18 June 2021, Athens, Greece.

Deep Reinforcement Learning for Robot Batching Optimization and Flow Control

Max Hildebrand[a,b], Rasmus S. Andersen[b], Simon Bøgh[a,*]

[a]*Robotics & Automation Group, Department of Materials and Production, Fibigerstræde 16, Aalborg, 9000, Denmark*
[b]*Marel A/S, Århus, 8100 , Denmark*

Abstract

Robot batching is an optimization problem found in many industrial applications. Current state-of-the-art approaches utilize a combination of heuristic based parameters and statistical analysis. This approach necessitates many tunable parameters, which again provides challenges when delivering systems to new customers. We challenge current state-of-the-art in statistical approaches by presenting a novel application of a policy gradient method for a Deep Reinforcement Learning (DRL/RL) agent. We have developed a Unity simulation framework of an existing robot-batching cell, on which a RL agent is able to successfully train and obtain a policy for performing robot batching, using a tabula rasa approach. The trained agent is capable of packaging 47.86% of 1218 total batches within the prescribed tolerances, with a positive give-away of 8.76%. The application of DRL in performing robot batching is to the authors knowledge the first of its kind.

Keywords: Robot Batching; Artificial Intelligence in Smart Manufacturing; Proximal Policy Optimization; Deep Reinforcement Learning

1. Introduction

Batching is a term used in industrial food processing to describe the act of combining smaller pieces into batches of a certain size.

The batching solution considered in this work is a two robot, aligned in the direction of an item introducing conveyor belt, robot batching cell. Batching is a task that requires a high amount of combinatorial accuracy in order to minimize give away and fulfill order requirements. Not only is the demand for accuracy high; the distribution of and set of weights is unknown in advance, as the items are continuously introduced and moved through the batching cell. The batching algorithm has limited time to estimate a best fit before it must assign a robot to perform the pick and place task. This adds significantly to the complexity of the task.

The partner company solves the task of batching by utilizing a patented probability-based method. During production, the weight distribution of the incoming items is estimated. This distribution is then used to calculate which items return the highest probability of achieving the target weight, when placed in a specific tray [6]. In addition to this, several other factors are considered alongside the probability estimate. In combination, a score is calculated of how well items match trays whilst also optimizing throughput.

A drawback of this method is that it is highly reliant on parameter tuning. This tuning is generally performed when first delivering a batching cell to a customer. This provides a statically tuned batching cell that works well, but over time it can start to perform sub-optimal as the weight distribution of product changes, as well as types of jobs, and other external parameters. This provides incentive for investigating new technologies that have the potential to replace current methods altogether or augment them to the point where there will be no need for readjusting parameters. The vision is to provide customers with a product that requires less service interaction and higher effective uptime. We hypothesize that Deep Reinforcement Learning (DRL) may provide one such solution, as it would only need to be tuned once in the training stage, and henceforth have the ability to adjust by generalization in the environment.

Since the release of the first Deep Q-Network (DQN) presented by Mnih. et al.[8], the field of DRL has seen an increase

* Corresponding author. Tel.: +xx xxxxxxxx ;
E-mail address: sb@m-tech.aau.dk (Simon Bøgh).

10.1016/j.promfg.2020.10.203

in both available off-the-shelf algorithm implementations and applications of increasing complexity. Although the vast majority of Reinforcement Learning (RL) applications pertain to training agents to play games, the underlying idea of performing trial-and-error in a feedback loop, e.g. the classic RL loop seen in Figure 1 in order to achieve a goal or score, can be applied in many industry applications.

In this paper we are inspired by prior research in the application of DRL in industrial automation systems. Andersen et al.. successfully trained an actor-critic RL agent in a brine injection process, where the goal was to minimize several parameters of meat curing[11]. In another paper, Blad et al. developed an RL agent capable of controlling a heating, ventilation and air-conditioning simulated environment[3].

In this paper we seek to further the research of industry applications of DRL, by building a simulation of a batching production setup and training a RL agent to solve the task of batching.

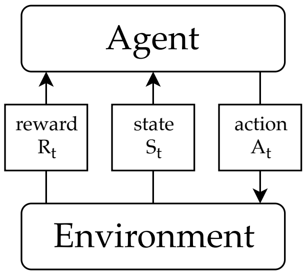

Fig. 1. The classic feedback loop of reinforcement learning

For an RL algorithm to be feasible in an industry process, the following key elements need to be present:

1. A feedback loop in the manufacturing process, which provides information about the state of the system before and after an action has been performed. The action can e.g. be pick-and-place.
2. A method for generating large amounts data,
3. or a large database of previously recorded data that can be labelled into states and actions.
4. Training environment for the agent to do trial-and-error while exploring the best policy.

The first element is inherently present in robot batching. The second and fourth are attainable by constructing a simulation that follows the logic of the real process. We therefore recognize the potential of investigating RL in the batching scenario.

2. Related work

The vast majority of RL implementations to date, are focused on playing games. Games serve as a good benchmarking tool, as the performance often can be compared directly to human performance. From playing 70 & 80’s 2D Atari games[7], to more complex games such as DOTA2[10], Starcraft [17] and GO [15], the goal has always been to reach and supersede human performance, which has been achieved time and time again. The first to achieve human performance in a videogame using DRL was the team behind DeepMind in their work ”Human-level control through deep reinforcement learning”[8]. They achieved this utilizing the off-policy method Deep Q-learning, and along the way invented several ML-methods that have since been applied in many newer RL-algorithms. Deep Q-learning was the first DRL algorithm, and since its publication the number of available reinforcement learning frameworks, algorithms and applications have increased significantly.

Silver et al.. solved the game of GO by developing a novel monte carlo simulation method augmented by RL trained on data derived from expert humans[15]. OpenAI et al.. hypothesised that current state-of-the-art pure RL algorithms have untapped potential, and proved it by applying Proximal Policy Optimization (PPO)[14] to the computer game DOTA2; a game with a considerable complexity [10]. When discussing the feats of AlphaGO (DeepMind’s GO agent), finding an optimal solution in the vast state-space is usually the highlight. In DOTA2, searching the state-space for an optimal next action would be near impossible, not only due to the vastness of the search space, but also due to time constraints, as the game is not turn-based as opposed to GO. Additionally, DOTA2 was solved by providing floating point inputs of various game information to the neural networks, proving that a complex environment can be reduced to a meaningful set of floating point numbers, allowing for a simpler network architecture than the Convolutional Neural Networks used to play Atari from a set of images[10].

Whilst the bulk of research is focused on playing games, recent years have also brought insights into possible industry applications. One such example is in [12], where Deep Q-Learning is applied to process control. It is demonstrated that DRL is capable of obtaining a policy that can replace the need for controller design. Controller design would under normal circumstances require manual tuning of parameters, similar to robot batching, and this result is therefore particularly interesting in regards to the current work. The authors hypothesise that the RL algorithm PPO, developed and applied by OpenAI et al.., can be utilized in solving the task of end-to-end batching. This is promising due to similarities in regards to selecting an appropriate action when considering a long term strategy whilst in a time constrained environment and reducing a complex environment to an input array of floating points.

3. Framework and Setup

In order to apply Proximal Policy Optimization to the robot batching problem, a simulation environment capable of representing the problem is developed. The simulation consists of the respective CAD models of the partner company’s robot batching solution and is designed using Unity [16]. The physical batching cell consists of the following parts:

- Weighing station
- Infeed conveyor

- Two tray feeding stations
- A left and a right side tray feeding conveyor
- A left and a right side tray lane conveyor
- Two Delta Robots
- Metal casing for the entire cell

For the simulation, the batching cell is modified by omitting the metal casing and by merging the tray feeding conveyor with the the tray lane conveyor. A screenshot of the Unity environment can be seen in Figure 2 and a descriptive overview of the simulated robot batching cell can be seen in Figure 3.

The simulation obeys by the following heuristics:

- Each item is assigned a weight drawn from a normal distribution
- Items are spawned with 1 second intervals
- Items are centered on the infeed conveyor, with identical orientation
- Infeed conveyor speed $0.5\frac{m}{s}$
- Tray lane conveyor speed $0.2857\frac{m}{s}$
- Tray lane conveyor advancement step size of 30cm
- Tray lane conveyor advancement occurs when possible, i.e. when trays that will leave the cell meet or supersede the target weight
- It is impossible to place in a moving tray
- Trays are of size 25x20x5cm
- Items are of size 10x5x5cm

In the real batching cell, the infeed conveyor is capable of different speeds depending on the job and configurations of connected machines. In the simulation, it has been set to a fixed speed, of $0.5\frac{m}{s}$. For the tray conveyors, the speed is $0.2857\frac{m}{s}$. This speed has been chosen to combine ans simplify several factors, including pre-movement unavailability, movement, post-movement unavailability. The tray conveyors advance in steps of 30cm whenever they are able to, without moving unfinished trays outside of the work area of the finishing robot. Both Delta robots can reach within a radius of 0.6m of the base center and move at a speed of 3.3m/s with infinite acceleration. It should be noted that depending on the position of the end-effector, items can be deemed unreachable even though they are still within the reachable space. This prevents the robot from moving towards an item that will have left the reachable space before it can be reached.

The simulation environment, seen in Figure 3, is reduced to an array of floating numbers, representing only what is considered of highest priority when assessing the batching process. A large battery of experiments lay the basis for distilling the input array to the following:

- Current Weight of reachable trays, at most 16
- Weight of up to 9 items on the infeed conveyor, where the first and second inputs are reserved for the item that will be interacted with should the front or finishing robot perform an action, the remainder serving as an information buffer
- Position of up to 9 items, following the same logic as the point above, provided in a single coordinate on the axis of movement of the conveyor
- 4 Boolean inputs representing the availability of either robot and both tray conveyors
- 2 Boolean inputs representing if either tray lane requires more than 1 tray to finish, before an advance can occur

3.1. RL Agent

The RL task of batching requires the agent to obtain a policy that can match incoming items by weight to obtain a given target weight, while at the same time ensure that trays move out of the batching cell. The RL agent proposed in the current paper utilizes the PPO algorithm. PPO is an on-policy RL method that builds on the idea of Trust Region Policy Optimization [13]. It utilizes the policy update constraint $r_t(\theta)$ and implements a clipping factor, which results in a novel objective function seen in equation 1, developed by Schulman et al.[14]. Schulman et al. attempted to ensure that no policy update step is too large. This guarantees to some extent policy improvement, albeit there can be no guarantee against local minima.

$$L^{CLIP}(\theta) = \hat{E}_t\left[min(r_t(\theta)A_t, clip(r_t(\theta), \hat{1} - \epsilon, 1 + \epsilon)\hat{A})\right] \quad (1)$$

The environment described in the previous chapter has a time step resolution of 0.02 seconds. This in turn provides the learning agent with resolution of 1cm per time step on distance traveled by items on the infeed conveyor. This again means that for every centimeter of item movement, the agent will have to assess state and attempt an action. The agent can in this environment perform a total of 18 actions at each step:

- Place item available to robot 1 in any of the 8 trays within its reach (8 actions).
- Ignore the item currently available to robot 1 until it becomes available to robot 2. This will move up the next item in the buffer to the actionable input of robot 1 (1 action).
- Place item available to robot 2 in any of the 8 trays within its reach (8 actions).
- Do nothing (1 action).

The availability of these actions depend on the state of the environment. As previously mentioned, information regarding when certain actions are possible are passed along the remainder of state information. So not only does the agent have to learn a policy that will minimize give away and ensure that trays are moving through the cell; it also needs to learn when certain actions are possible and not. In addition to this; tray advancement configuration presents a challenge in the sense that in certain tray placement configurations, the heuristics of the environment call for the two front trays to be filled within limits for an advancement to occur. As this is a rarely occurring phenomenon, it presents a considerable learning challenge for the agent.

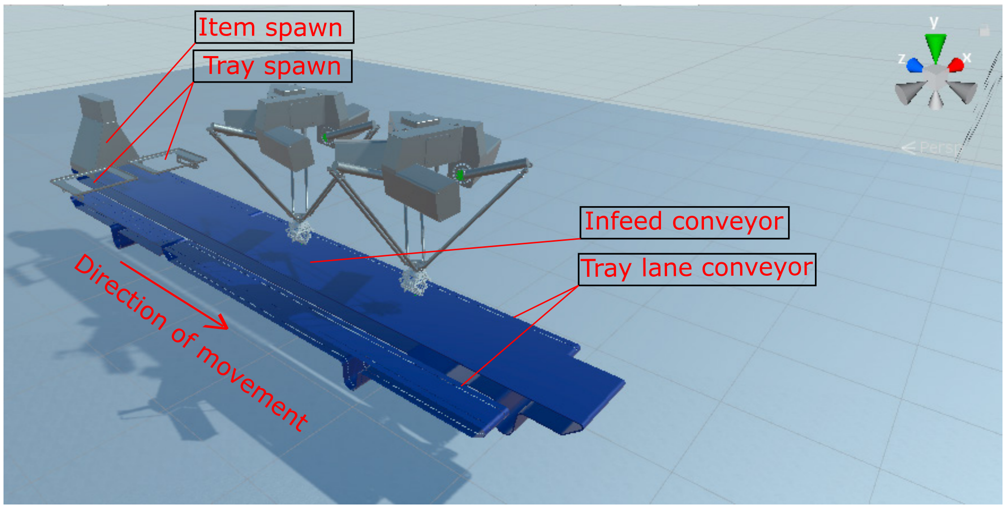

Fig. 2. Simulation environment developed in Unity. The simulation environment contains two parallel robots placed over a conveyor system. On either side of the center conveyor, empty trays are continuously fed for the batching process where objects are filled in. On the center conveyor, new objects are continuously spawned with a given weight distribution. All item and tray movement is from the point of view, left to right.

4. Experiments and results

4.1. Neural network configuration

When choosing the network architecture for a new RL application; one with few or no predecessors, a multitude of rule-of-thumb approaches exist for where to begin. One way is to base the initial design on intuition. Throughout the development of the simulation, an RL agent designed on intuition has been introduced each time a milestone had been achieved. This has provided ample chance to test various network settings. Arriving at the final simulation environment, it has been possible to test the various network architectures found throughout development. In Figure 4, a comparison between the three most successful network architectures can be seen, and it is clear that the simplest network architecture not only learns faster, is more stable but also reaches a better policy. All tested networks utilize ReLU neurons[1].

It should be noted however, that the network architecture performance is extremely dependent on the hyperparameters *Horizon* and *mini-batches*. Hyperparameters are discussed in the next subsection.

4.2. Hyperparameters

Like all other RL algorithms, PPO has several hyperparameters that can all impact the learning agent in different ways. The hyperparameters and their settings for this agent are:

It was found that horizon and mini-batches are most significant for the performance and also most challenging to tune. Horizon determines how far we look into the future in regards to the sum of rewards used for the policy gradient update. Mini-batches should cover the steps from an action has been taken until a reward has been provided. As such, the complexity of a strategy, that a policy can learn is determined by the size of these parameters, most notably the horizon.

Table 1. Hyperparameters for the experiment.

Hyperparameter	Value
Discount (γ)	0.99
Entropy coeff.	0.0001
Learning rate (α)	0.00025
Horizon (T)	2048
Mini-batches	16
Clip range (ϵ)	0.2
GAE Lambda(λ)	0.95
Value loss weight	0.5

This simulation environment reaches approximately 82000 steps pr. 1000 items batched. In comparison to the DOTA2 PPO implementation, where complex strategy is learned, they scale the environment to ultimately provide 80000 time steps per game finished. They distribute the training between multiple systems; obtaining 60 batches of size 1048576 per minute to train on[9]. An additional key feature of the DOTA2 implementation is the utilization of Long Short Term Memory (LSTM), which greatly increases the NN complexity but allows for more coherent strategies.

4.3. Reward Function

The reward function is the core of the agent. In order to obtain any kind of meaningful behavior, a valuable target for optimization must be established. The desired behavior of the agent, is to pair items to reach a target weight, but also ensure trays leave the cell. For the placement of items into a tray, the reward is based on the deviation from the target weight:

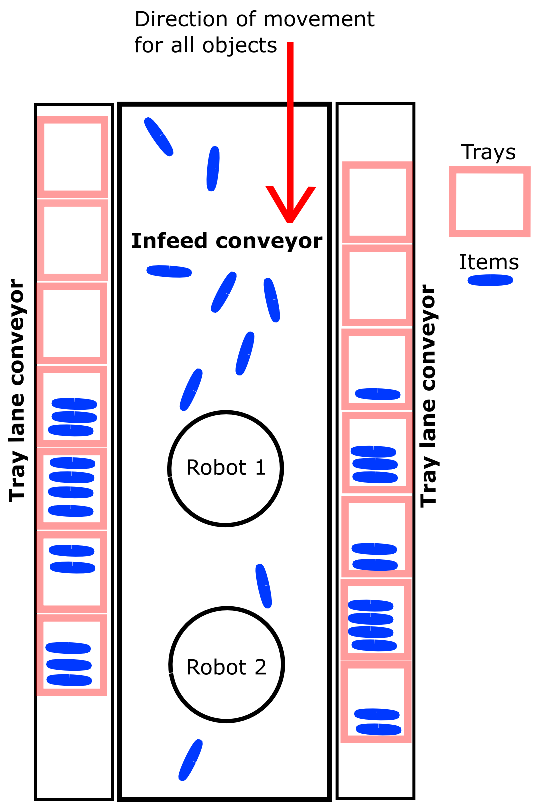

Fig. 3. Overview of the robot batching cell, with affixed names. The infeed conveyor moves items forward towards the two robots placed above the conveyor. The tray line conveyor provides new empty trays to be filled during the batching process.

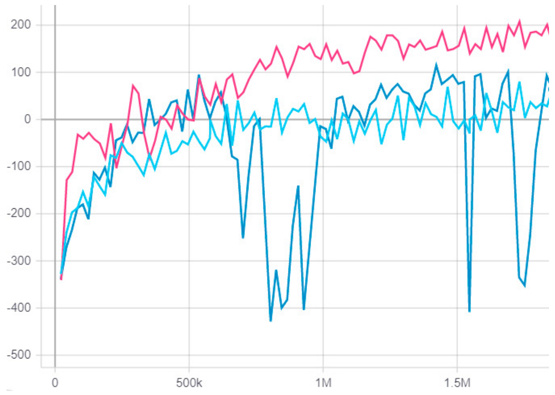

Fig. 4. A comparison between network architectures; [128,128] dark blue, [64,64] light blue, [32,32] pink. It is clear that the first of the aforementioned networks quickly runs into overfitting, whereas the latter is stable throughout and reaches a better policy.

- new weight < target weight - tolerance
  reward = $\frac{4}{newweight}$
- $\sqrt{(newweight - targetweight)^2}$ < tolerance
  reward = 0.15
- $\sqrt{(newweight - targetweight)^2}$ < 10g
  reward = 0.2
- new weight = target weight
  reward = 0.25
- new weight > target weight + tolerance
  reward = -0.10

Additional terms:

- Item passes all the way through the cell
  reward = -0.14
- finished tray leaves the cell within tolerance
  reward = $\frac{2}{\sqrt{(trayweight-targetweight)^2}}$
- finished tray leaves the cell with less than 10g give away
  reward = 0.2

The reward function is designed to give a strong signal when the agent either hits the target weight or comes close within the established tolerances of the batching job. For this particular scenario, the tolerance has been set to 30g either positive or negative deviation from the target weight. The very first term, $\frac{4}{newweight}$, provides an increasingly stronger signal, the closer the weight gets to the specific tolerance used. At 31g deviation, the reward is 0.12, and at 30 it becomes 0.15. Thereafter the reward can at most increase by a factor of 1.6, given the agent hits either less than 10g or no deviation at all, as seen in Figure 5.

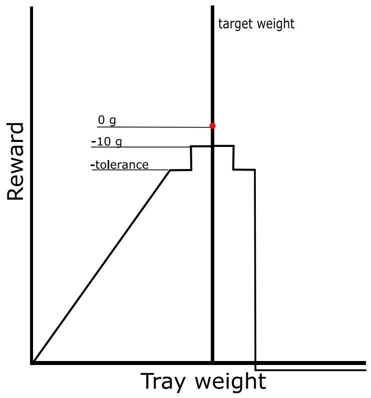

Fig. 5. The reward function. The reward linearly increases up until the batch weight reaches within tolerance. This provides incentive to match items to reach as close as possible to 0 g give away, which releases the strongest reward. Any action that leaves a batch with a weight that supersedes the positive tolerance, releases negative reward.

The additional terms provide some incentive as to avoid letting items leave the cell and for pushing trays out of the cell, but only rewards those who leave the cell within the tolerance. If the aforementioned mini-batch and horizon hyperparameters could be increased significantly, this reward term would gain a greater influence on the learned policy.

4.4. Training

Training of the agent has been conducted using Unity ML-Agents[5], stable-baselines[4] and the OpenAI Gym API[2].

The agent has been trained for upwards of 30 million time steps, where one episode consists of approximately 33000 steps and 400 items batched, corresponding to batching approximately 363600 items. It is however found that improvement stagnated beyond 4 million steps. In order to extract the best performing policy, it is evaluated on an average reward return across 40000 steps, upon each concluded episode. During training runs, the various policies tested are compared to a baseline test. In the baseline test, all items spawned weighed exactly $\frac{1}{4}$ of the target weight. This makes the learned policy as simple and easily obtainable as possible. Of all the training runs performed, the final agent represents the policy that came closest to a perfect score compared to the baseline test.

4.5. Results

As a final test to evaluate the batching performance, the best agent extracted from training has been inserted into a test environment requiring 4000 items, generated from a normal distribution with mean 200 and a standard deviation of 28, to be batched. This is the same type of distribution on which it was trained. The distribution of the generated item weights can be seen in Figure 6. The batching job itself is prescribed as target weight 600g, tolerances ±30g.

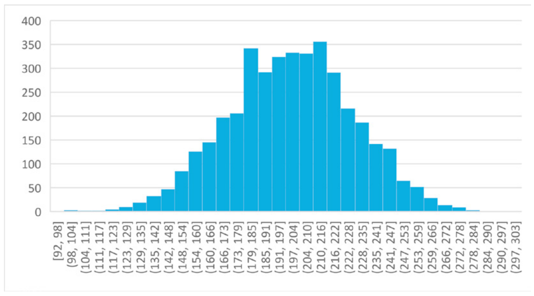

Fig. 6. Histogram representing items introduced in the batching cell during the final test. Mean 200, standard deviation 28.

In total, the agent finished 1218 trays, 573 of which were within the set tolerances of ±30g. A success rate of 47.86%. In Figure 7, the distribution of the finished batches can be seen, and it can be noted that upwards of 100 trays were less than 1 standard deviation away from being accepted. The agent placed an average of 3.28 items into each tray, and obtained an average tray weight of 654.48g, with a standard deviation of 65.13g. The agent batched 100% of items.

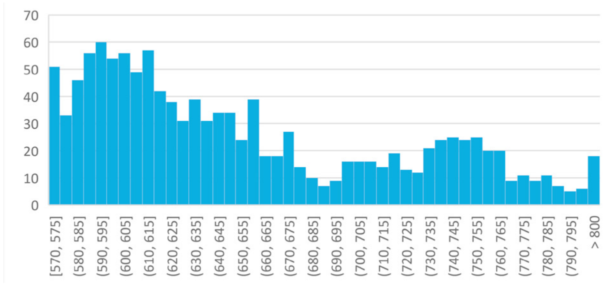

Fig. 7. A histogram of the batches produced by the final agent. Everything between 570 and 630g is considered a successful batch.

In order to compare the agent with the state-of-the-art, a simulation tool from the partner company was used. The simulation tool utilizes the statistical batching method, and the exact same batching cell layout as the RL agent. It was fed the same 4000 items and asked to perform the same batching job. In total, it finished 1294 trays, 1294 of which were within the aforementioned tolerances. The average tray weight is provided by each individual tray lane, as opposed to the RL agent combining it, and adds up to 600.12g and 600.45g. Opposite to the RL agent, which batched 100% of items, 103 items were discarded during the batching job.

5. Discussion

The application of the PPO agent in the batching environment shows promise. The agent is able to successfully learn a policy. It overcomes the challenge of double tray advance and batched trays without producing excessively gross overweight. As mentioned previously, a correlation between the hyperparameters horizon and mini-batch size and the learned policy is still to be thoroughly investigated. The primary reason for the values attributed to the aforementioned hyperparameters is the fact that training time required for convergence grows considerably with these parameters; most notably horizon. With the hardware available for conducting training of the agent in the current work, the parameters had to be kept small.

It was noted during testing that the agent utilizes only the last robot. Also, at most 2-4 trays are being filled at any time. This showcases a rather shortsighted strategy, albeit successful. This shortsighted strategy clearly indicates a necessity for increasing the horizon parameter in future work. In addition, the reward term which rewards trays leaving the cell further enforces the shortsightedness of the strategy of only utilizing the trays close to exiting the cell. This term should be revised in future work to provide less incentive to focus on the finishing robot, regardless of the horizon parameter. Having a mini-batch size of 16, corresponds to observing the steps of taking an action; having the robot move the item into the tray and receive the reward. This is considered of reasonable size.

6. Conclusion

The hereby paper presents a novel application of a Deep Reinforcement Learning algorithm in a simulated robot batching environment developed in Unity. The final batching job performed by the trained agent, results in 47.86% successful batches, clearly demonstrating that a DRL agent is capable of learning the task of robot batching with a tabula rasa approach. We hypothesize that with more powerful hardware and additional hyperparameter tuning, a significant increase in acceptable batches can be achieved. Future work will focus on increasing the batching accuracy, i.e. increasing the successful batches and decreasing give away. The approach shows to be very promising in the domain of robot batching and it is believed that it can provide similar or better batching results than existing statistical methods.

References

[1] Agarap, A.F., 2018. Deep learning using rectified linear units (relu). arXiv:1803.08375.

[2] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. Openai gym. arXiv:arXiv:1606.01540.

[3] C.Blad, S.Koch S.Ganeswarathas C.S.Kallesøe, S., 2019. Control of hvac-systems with slow thermodynamic using reinforcement learning. FAIM 2019 38, 1848.

[4] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., 2018. Stable baselines. https://github.com/hill-a/stable-baselines.

[5] Juliani, A., Berges, V.P., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D., 2018. Unity: A general platform for intelligent agents. arXiv:1809.02627.

[6] Kvisgaard, T., Bomholt, J., 1996. Patent number wo1996008322a1: Method and apparatus for weight controlled portioning of articles having non-uniform weight.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv:1312.5602.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

[9] OpenAI, . Openaifive, our team of five neural networks, openai five, has started to defeat amateur human teams at dota 2. https://openai.com/blog/openai-five/.

[10] OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H.P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., Zhang, S., 2019. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680.

[11] Rasmus E.Andersen, Steffen Madsen, A.B.S.B.M.N.R.S.S.B., 2019. Self-learning processes in smart factories: Deep reinforcement learning for process control of robot brine injection. FAIM 2019 38, 1848.

[12] S. P. K. Spielberg, R.B.G., Loewen, P.D., 2017. Deep reinforcement learning approaches for process control. 2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP) .

[13] Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P., 2015. Trust region policy optimization. arXiv:1502.05477.

[14] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv:1707.06347.

[15] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. nature 529, 484.

[16] Unity Technologies, 2020. Unity game development software. URL: https://unity.com/.

[17] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D., Silver, D., 2019. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/.