Batching using reinforcement learning

M. Hildebrand, J. Frendrup, M. Sarivan

Department of Materials and Production, Aalborg University
Fibigerstraede 16, DK-9220 Aalborg East, Denmark
Email: mhilde15@student.aau.dk,
Web page: http://www.mechman.mp.aau.dk/

Abstract
When performing batching, the governing goal is to keep give-away at an acceptable level. Giveaway being defined as weight that is added or absent from a given target weight. The product being packaged varies in dimension and weight, and as such it presents a challenge when matching multiple products to fit a target total weight. The current batching setup uses a cell consisting of multiple robots and multiple batching packages, with a product conveyor in the middle and a package conveyor on each side, moving independently. The package can only leave the cell by meeting the weight requirement. This presents additional challenges, as there is a risk of packages at the beginning of the conveyor reaching the weight requirement long before it can leave the cell, and thus the cell runs the risk of being unable to batch. Currently, this is solved by using strict distribution keys, coded into the system, making it highly inflexible to changes in the product weight distribution. In order to avoid down time, increase flexibility, remove the strict distribution keys and minimise giveaway, this project will investigate the implementation of a reinforcement learning algorithm to achieve the aforementioned goals.

Keywords: Reinforcement learning, Batching, Deep Q-learning, Artificial Neural networks

1. General Introduction

When performing batching of products that have varying weight, the primary goal is to achieve a target package weight with minimal deviation. In an effort to achieve the goal weight, companies utilise rigid distribution rules that require tuning for each specific batch, with different weight distributions. In an effort to reduce the setup time for a new batch, and possibly further reduce giveaway, this article investigates the implementation of reinforcement learning (RL) in a batching environment.

Moreover, an imprecise batching process can lead to both positive and negative giveaway. The positive giveaway is what the customer gets in excess, the negative giveaway is what the customer is losing. To protect the consumers from buying under-filled products, the European Union Directive has made a legislation that regulates the tolerable negative error. This is the allowable amount with which a product can fall under the weight specified on package. This legislation is known as e-weighing [1].

The specified requirements, acquired from the European Council Directive legislation [2], defines considerations for acceptable batching. This is acceptable if the mean value

$$\bar{X} = \frac{\sum X_i}{n}$$

of the actual contents X_i of number n packages in a batch is greater than the value:

$$Q_n - \frac{s}{\sqrt{n}} \cdot t_{(1-\alpha)} \tag{1}$$

where the indicated weight for each package, called Nominal Quantity, is denoted Q_n and $t_{1-\alpha} = 0.995$ confidence level of a Student distribution with $V = n-1$ degree of freedom. The value of the standard deviation s of the actual contents of the batch can be estimated by the following calculations:

The sum of the squares of the measured values:

$$\sum_{i=n}^{i-1}(x_i)^2 \tag{2}$$

The square of the sum of the measured values:

$$\left(\sum_{i=n}^{i=1} X_i\right)^2 \tag{3}$$

Then:

$$\frac{1}{n}\left(\sum_{i=n}^{i=1} X_i\right)^2 \tag{4}$$

The corrected sum

$$SC = \sum_{i=n}^{i-1}(x_i)^2 - \frac{1}{n}\left(\sum_{i=n}^{i=1} X_i\right)^2 \tag{5}$$

The estimated variance:

$$V = \frac{SC}{n-1} \tag{6}$$

The estimated value of the standard deviation is:

$$s = \sqrt{V} \tag{7}$$

It is desired through the development stage presented in this article to achieve a viable RL solution that can be implemented in a product batching scenario and that it is able to abide by the boundaries set by the e-weighing regulations without gross positive giveaway.

2. Example batching method

One batching method, patented by Scanvaegt A/S, calculates the probability of occurrence of the products weight, and is done so with a continuously updated normal distribution of the products entering the system. This batching method builds on top of the *accumulation weighing method* [3], which is a simpler method of batching and used when products are moving on a conveyor leading to an allocation machine. The products are weighed on a dynamic weigher and then placed into selective bins for build-up of portions, until the bin exceeds the target weight, and are discharged into trays. It is generally understood that the last product discharged to the bin, which is to the bring the total weight of the portion to the desired target weight, will bring the bin above the target weight, resulting in giveaway, as it will be an almost lucky coincidence if the arriving new product will fit the portion weight precisely. Furthermore, to try and correct the overweight error, the accumulation weighing method selectively match the above and below average product-weights to a certain degree of success[3].

The batching method patented by Scanvaegt, builds on top of the accumulation weighing method by predicting the weight of the arriving products with a more accurate distribution. Said distribution is obtained by registering, the weight of previous (50-300) products entering the batching system with the use of a "serial register" using the type first in, first out (FIFO), to form a histogram, and is continuously updated to eventual shifts in the incoming supply. With this adaptive and fine-tuned representation of the distribution it is possible to calculate a probability function, with higher certainty, for the occurrence of new products and their ability to fit partly filled trays. Moreover this method makes the pairing of deviating products more precise. [3]

An example of this, could be when placing the first product into one of the empty trays in the physical setup. This placement can be done uncritically, but each subsequent product that is to be placed will be evaluated in regards to how the placement will affect the probability function in respect to successfully filling the tray closer to the target weight. The probability function of the two products is derived by multiplying their respective probabilities together, to form a specific weight, where each probability is given from the probability density function. Calculating the probability for three or more follow the same procedure. With these calculations it is possible to have a representation of all the probabilities for the summed up products having different weights.[3]

It is important to know said calculations are not directly utilised for batching to the desired target weight, but rather for calculating the probability function of the weight for the number of missing products, backwards from the target weight to zero, in all trays, as this leads to better placement of the products.[3]

Additionally undisclosed parameters are adjusted as well to increase precision of the batching algorithm.

A way to decrease the number of parameters which are to be tuned when adjusting to the eventual shifts in the weight distribution of the products would be to utilise RL, that would be able to adjust without human intervention or manual tuning at any point, to changes in the product distribution or new production environments.

The remainder of this article, will explore the implementation of RL algorithms in a fictive manufacturing environment.

3. Reinforcement Learning

RL is the introduction of a learning agent in an environment, seeking to optimise reward, based on actions taken. In this article, the particular RL method applied is known as Q-learning. In Q-learning, an update based adaptation of the Bellman equation, seen in equation 8, where the Q-value to a state, given an action, is expressed through an immediate reward, and a prediction of future Q-values.

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma(\max(Q(s',a') - Q(s,a)) \tag{8}$$

Where $Q(s,a)$ denotes Q-value for the state, s, given action a. α is the learning rate parameter, r is the

immediate reward, $\max(Q(s', a'))$ is the maximum predicted Q-value, given the next state, s', and the action that will lead to said maximum Q-value, s'.

Q-learning utilises the epsilon-greedy policy. In essence, this is the separation between choosing actions based on exploration, random actions with a probability of ϵ, or choosing actions believed to give a better outcome, greedy actions.

In order to implement RL in the batching setup, the environment has to be converted to a model that can be progressed in episodes or time-steps. This, alongside an increase in the complexity of the learning algorithm, is done during several iterations. In iteration 1, Q-learning in a finite environment and brute force algorithms were explored. In iteration 2, the environment is expanded and consequently the Q-learning algorithm as well, deep Q-learning is introduced. In iteration 3, the environment is expanded further, and methods from Mnih et. al. are implemented to increase the performance[4].

4. Development

4.1 Environment definition

As mentioned, the environment has to be converted into a software structure.

- **The conveyor belt** that transports the product into, and out of, the packaging cell.
- **The products** are the items needed to be packaged inside the cell.
- **The trays** are the packages, in where the products are to be placed.
- **The buffer** represents a variable number of products, that the system will could be able to see before being able to interact with.
- **The target weight** is the weight of a tray at which giveaway is 0
- **The robotic manipulator** which will be referred to as the agent.
- **The giveaway** is the extra weight or the lack in weight a tray has when it is deemed finished.
- **The distribution** of the products with a given mean and span.

The first 4 items in the list, are the state in any given time-step, presented to the learning agent in the shape as $[T1, T2, [VP_1, VP_2, VP_3, VP_4]$[Buffer]], where T1 and T2 are tray 1 and tray 2 respectively, VP_n is visible product, i.e. the agent can interact with it, and buffer represents the products that can be seen but not interacted with. All data is presented as integer values. Of the remaining 3 points on the list, one is the agent itself, one is the basis for reward and the last one a hidden parameter of the environment.

It should be noted that even though the buffer was ready for implementation and testing, the iteration process did not progress far enough for it to be implemented.

4.2 Tuning parameters

In the various iterations presented in this article, several tuning parameters exist. These include, but are not limited to:

$$\epsilon_{\text{initial}} = \{\text{exploration rate}\} \tag{9}$$

$$\epsilon_{\text{decay}} = \{\text{exploration decay factor}\} \tag{10}$$

$$\epsilon_{\text{min}} = \{\text{min. exploration rate}\} \tag{11}$$

$$\alpha = \{\text{step-wise update rate}\} \tag{12}$$

$$\gamma = \{\text{weight of prediction}\} \tag{13}$$

4.3 Reward function

Throughout all iterations, the reward function is based on the giveaway of the individual tray, and can be seen in equation 14

$$r = \frac{1}{\sqrt{(\text{Target weight} - \text{trayweight})^2}} \tag{14}$$

This function, provides the value for the parameter r seen in equation 8.

4.4 Iteration 1

Iteration 1 inherits the state description from section 4.1, and as such a learning agent can interact with any product on the conveyor at any time-step. The environment consists only of four products, with the

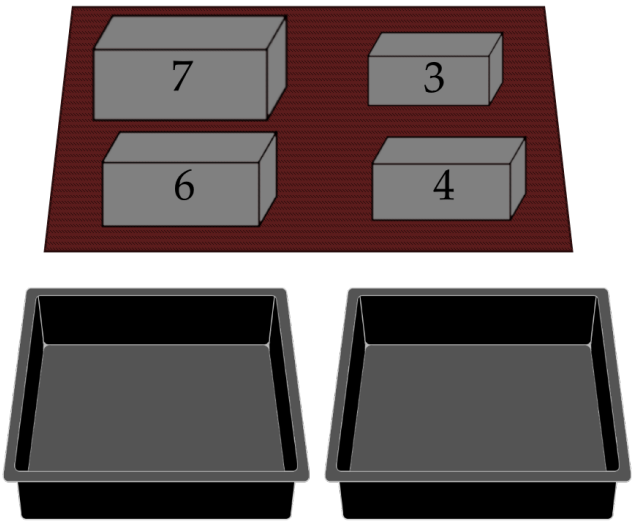

Fig. 1 Product batching scenario for Iteration 1.

weight values of $[4, 3, 7, 6]$, to be batched into two trays, each tray having a target weight of 10. Figure 1 showcases this particular scenario. The Q-learning method applied, utilises what is known as a Q-table, an example of which can be seen in table I, where T_1 refers to tray 1, and VP_1, as previously mentioned, visible product 1, and as such, an action is designated by the product and the tray into which it is placed.

State \ Action	$T_1(VP_1)$	$T_1(VP_2)$	$T_1(VP_3)$	$T_1(VP_4)$
25	-5	0.0	0.0	-0.1
26	0.0	-5	0.0	-0.1
27	0.0	0.0	0.0	-0.1
28	0.0	0.0	0.0	1

Tab. I An example Q-table.

The values within the Q-table are updated during an episode of training, using equation 8. In iteration 1, an episode of training consists of packing the 4 available products. The optimal solution of the environment, is an equal distribution of 10 in each tray, and this was achieved using the tuning parameters shown on figure 2. The plot also shows the convergence towards zero giveaway, and that a stable Q-table had been achieved at approximately 272 time-steps.

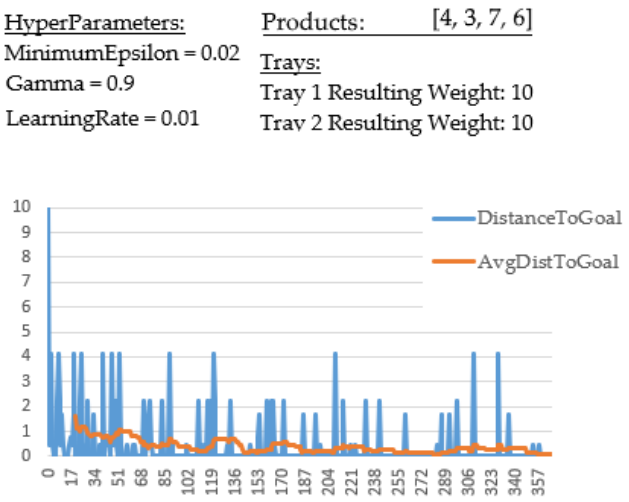

Fig. 2 The final result of iteration 1 with affixed tuning parameters.

4.5 Iteration 2A

In iteration 2, as previously mentioned, the complexity of the environment is increased. The complexity step in this iteration, is an increase of the product variance and the combinations of products that can be seen. On the agent side, there is also a change in complexity; in the real world, the machinery operates on a FIFO basis, and as such the environment model should reflect

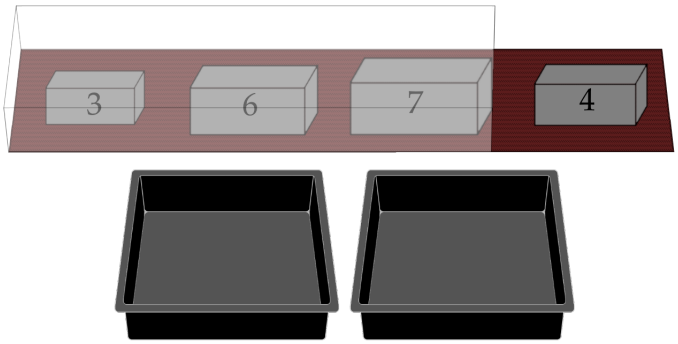

Fig. 3 Product batching scenario for iteration 2.

this. Therefore the action set is decreased. From this follows, the states presented to the agent in this iteration, take the form: $[T1, T2, [VP_1]]$, where the change is the amount of visible products, and an introduction of the non-visible products. The non-visible products are there, as the complete state still contains 4 different products, but are not shown to the agent. A visualisation of this batching scenario can be seen in figure 3.

As mentioned, the action set is decreased, from 8 to 2, as the action set is determined by $trays \times products$. The environment change is the primary challenge in this iteration. As the amount of possible products, and product combinations increase, as does the Q-table. From this follows an increase in training time needed. For the agent to visit all possible states, and update the Q-table enough times in order for a an algorithm to perform the actions needed to arrive at the optimal solution for each combination, the learning time-steps becomes increasingly large relative to the μ and σ of the product distribution. Therefore, as it can be observed in figure 4, the algorithm is highly unstable and does not converge. The yielded results are poor, since the trays showcased as an example in figure 4, do not meet the expected resulting weight. The poor results indicate the necessity of a different approach involving RL.

4.6 Iteration 2B

Having the Q-learning method proved to be ineffective when the weight of the products to be packed varies across episodes, it was replaced with Deep Q-Network (DQN). DQN is a combination of Q-learning and Artificial Neural Networks (ANN). Unlike Q-learning which makes use of a continuously increasing Q-table, DQN works by approximating the optimal action-value function. Value functions indicate how good it is for the agent to be in a particular state. This estimation is done though a Neural Network (NN) structure that in the hereby context maps weight values to agent

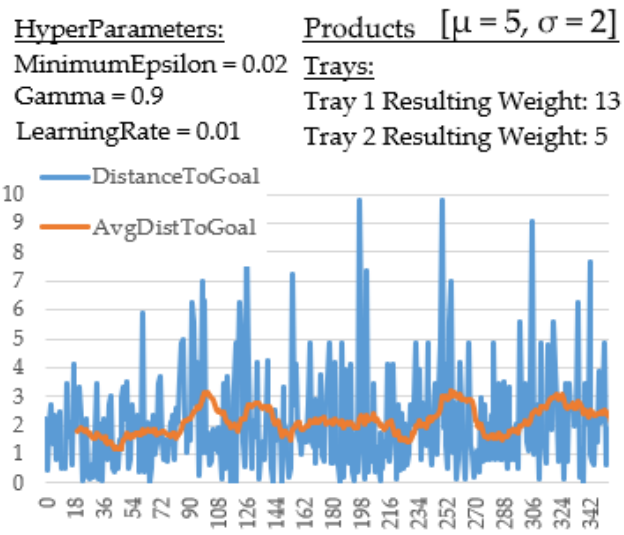

Fig. 4 Plot showcasing the behaviour of the Q-learning algorithm in the context of Iteration 2

actions in order to minimise the giveaway on packaged products. For designing the DQN, the following have to be considered:

1) Model Type: Originally, the ANN model type for DQN is the convolutional neural network (CNN). This model type however is common for image processing applications[4]. Therefore, the Multi Layer Perceptron (MLP) is selected for the product packaging application, as it offers flexibility in regards to input data[5].
2) Activation Function: as found by Krizhevsky et al. [6], the ReLu activation function can in a similar case as the one described in the hereby article lower the training time needed.
3) Loss function: as recommended in the loss function tutorial by Jason B. [7], it is chosen to be mean squared error (MSE).
4) Optimisation algorithm: as the defined problem by Iteration 2 is large in terms of data (various product weights), "Adam" was found to be the right choice for the hereby scenario [8].
5) Amount of neurons in the hidden layer (network width): subject to testing.
6) Amount of hidden layers (network depth): subject to testing.

In order to gain insight in the network architecture design, the depth and width of the NN, several tests were conducted. These tests consisted of training the network with different architectures. Firstly, the width of the network was tested, i.e. the amount of neurons in a layer. Secondly, the depth of the network was tested, i.e. the amount of layers. The initial results, for the distribution used in iteration 2, showed that a combination of 32 neurons and 2 layers performed best. However, an additional test was then conducted using the distribution that will be applied in iteration 3, in order to gain an understanding of the importance of design vs. distribution, and these results pointed towards an optimal architecture of 8 neurons, and two layers for this environment. The results of the tests can be seen in figure 5, where the aforementioned design with 8 neurons clearly stands out from the remaining configurations. In regards to the two layer design, the results can be seen in figure 6, where it can be observed that two and three layers outperform one layer, in regards to speed of convergence. As two and three layers eventually converge similarly, two layers is chosen to avoid future overfitting and unnecessary complexity.

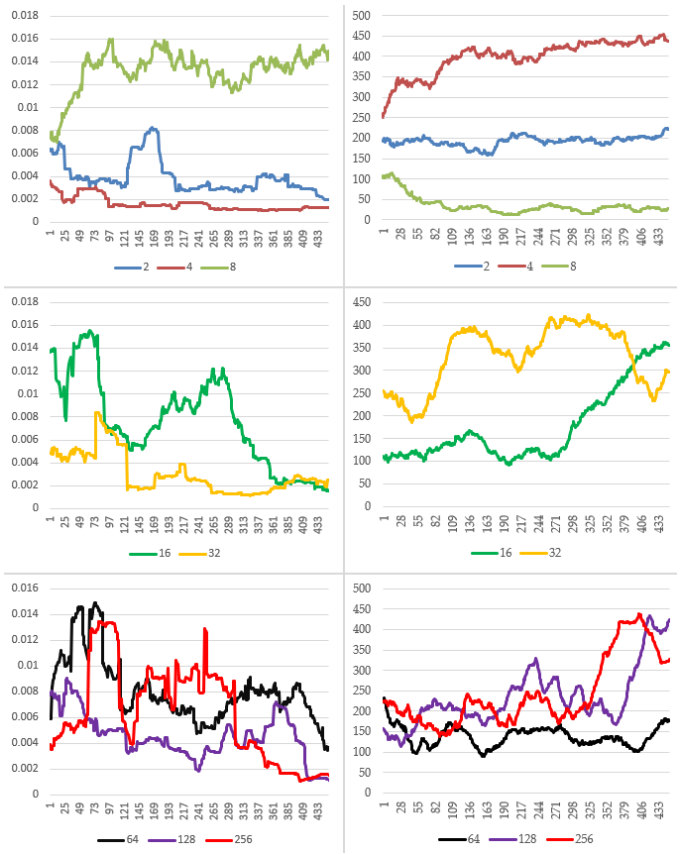

Fig. 5 Plot showcasing the behaviour of the Q-learning algorithm, with affixed neuron count. On the left side of the graphs, is the plotted reward, on the right the giveaway

4.7 Iteration 3

In iteration 3, the complexity is again increased. The environment up until now has consisted of two trays, and

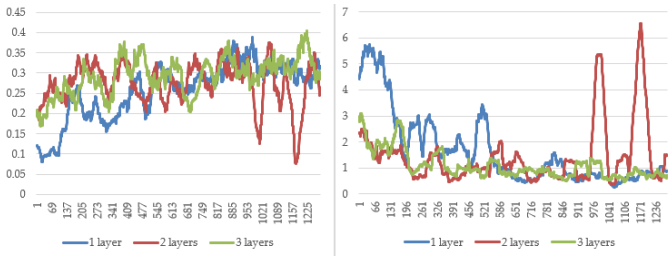

Fig. 6 Plot showcasing the behaviour of the Q-learning algorithm, with affixed layer count. On the left side of the graphs, is the plotted reward, on the right the giveaway.

episodes of 4 products, with a product distribution of, relative to the iteration 3 values, small μ and σ. In this iteration, the amount of trays is increased to 4, and the products per episode to 500. A substantial change in the iteration 3 environment compared to the predecessors is the product distribution. Now, μ and σ are increased to 253 and 43 respectively, providing a much wider distribution from which product weights can occur. Alongside these environment changes, an additional change, is the replacement of trays considered full, with empty ones. With these changes implemented the environment moves another step closer to a real production setup. The products per episode, or maximum time-steps parameter, is technically a tuning parameter, more than it helps mimic reality. In a real world setup, the batches are typically much larger, but the desirable learned behavior here, is hitting the target weight of a tray. As such, the maximum episodes is set to 500, in order to generate enough reward producing actions, actions that lead to trays being removed, for the replay and tuning of the network. The concept of replay, as just mentioned, is to gather experience rather than tuning on the spot, effectively decoupling the experience gathering and training. This method stems from Mnih et. al. who used it to gain *"Human-level control through deep reinforcement learning"*[4]. The two primary "tricks" employed in the article, is fixed target NN and experience replay. Experience replay helps in the sense that experiences can be used several times to tune the network, rather than just tuning the network once, upon having the experience. Fixed target NN decrease time spent learning, as the target Q-values stay stationary for a fixed amount of time-steps. With these methods applied to the DQN from iteration 2, the agent became competent within the now more complex environment. Additional architecture adjustments were however needed, due to the changes applied. These adjustments were performed based on tests similar to those performed during iteration 2B, and from this the final design of 32 neurons, 2 layers was chosen. The DQN was trained for 1.5 million time-steps, but gross overfitting occurred, and the best results were achieved after 100.000 time-steps. The reward and give away plot for the the aforementioned time-steps, can be seen in figure 7, where the average giveaway for a duration of approximately 100 episodes, 50.000 time-steps or products packaged, stays under 1.000. This is significant, as the giveaway is less than 8 per finished tray, following equation 15, which provides an estimate of the average giveaway per tray, if the episode average was 1000. The network weights and design that provided these results were then saved and used for the final tests.

$$\frac{1000}{\frac{\mu \times \text{timesteps}}{\text{targetweight}}} = 7.90 \tag{15}$$

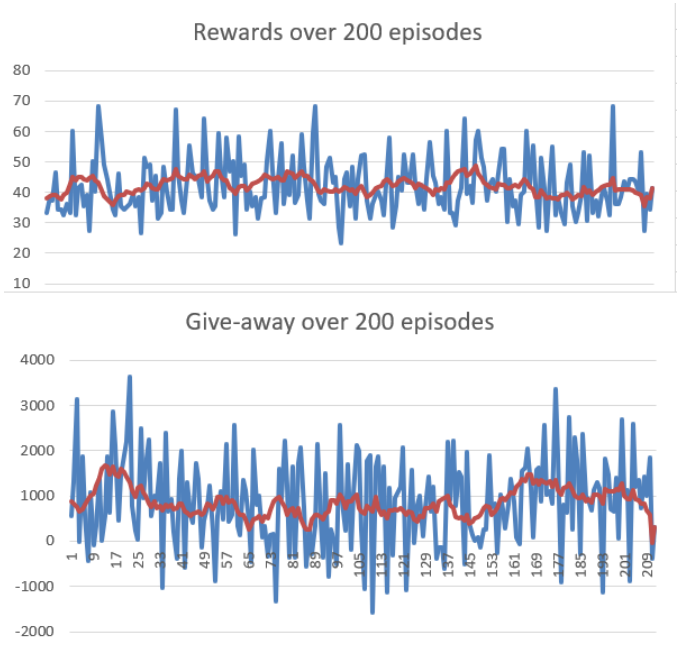

Fig. 7 The reward and giveaway plot for the final DQN.

5. Conclusion

The hereby article presented an approach to tackle the optimisation problem raised by product batching when there is a certain amount of variation in weight across products depending on a given normal distribution, and giveaway (goal weight overshoot/undershoot) is desired to be avoided. The iterative development process provided insights into Q-learning methods, in what scenarios they are beneficial to use (during iteration 1). Iteration 2 made the transition from Q-learning to the DQN method as described in Iteration 3 which ended up generating the following final results.

5.1 Test results

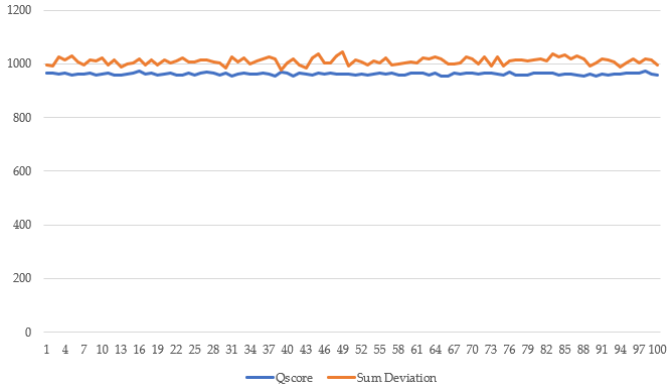

Fig. 8 A line graph displaying the e-weighing value and the mean of the sample. Accepted score is mean > e-weighing value

By using the DQN method, 100 episodes of 500 products were batched into ∼125 trays per episode. From each episode, a random sample of 30 trays is extracted, and used for calculating the score according to the e-weighing regulations[2]. The results can be seen in figure 8, which clearly show that all sampled trays pass according to the e-weighing regulations, forming the basis of the conclusion that the current environment setup of iteration 3 is sufficiently solved. The average giveaway during the testing was found to be ∼1000g, and considering that each episode consists of approximately 126 finished trays, making the average giveaway equal to less than 10grams per tray, per episode, and can be seen in figure 9.

5.2 Future work

The most complex batching problem defined during the development process is in Iteration 3. The problem definition however, only captures the details of a real-life batching scenario at a conceptual level only. It is natural to expect that the developed algorithms might behave differently or unexpectedly when applied on a system composed of:

- A conveyor feeding in the products
- The movement of the conveyors feeding in the trays
- The possibility that some products might not fit the target weight goal set for trays
- The position of the batching robotic manipulators
- The reachable space of the robotic manipulators
- Sudden changes in the weight distribution of the products

All these elements have to be considered in future development. Also, if the iterative approach is kept, this

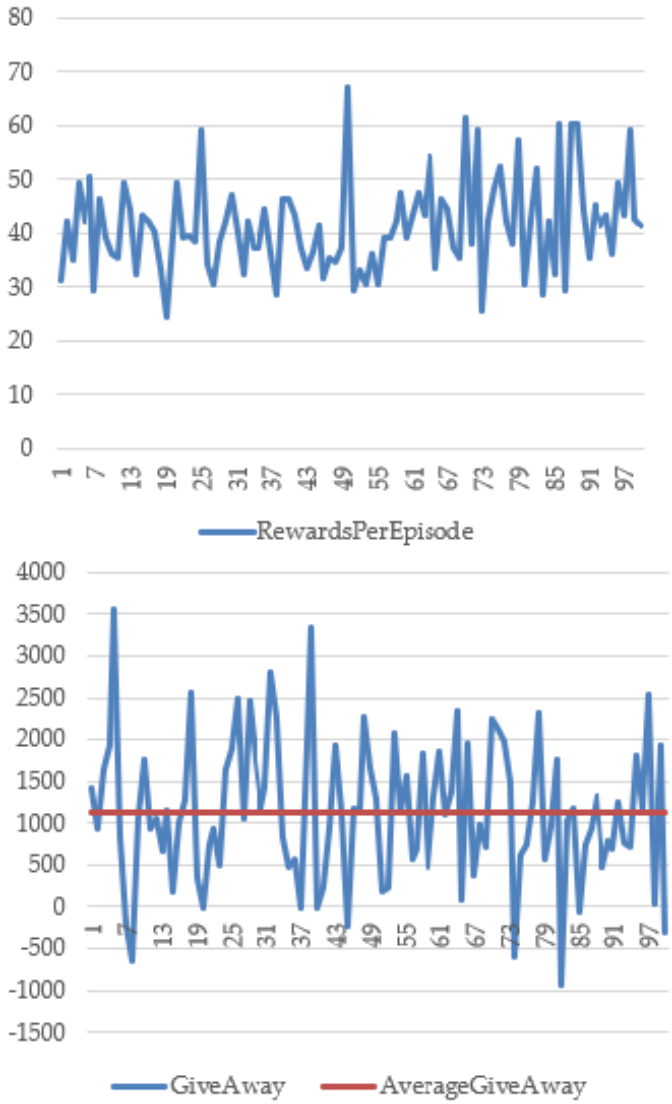

Fig. 9 The reward per episode and the giveaway during the iteration 3, final DQN test.

list above can serve as guideline for organising further iterations, from the system's complexity perspective.

In addition to Q-learning and DQN methods, there are also policy-based algorithms. These algorithms model a direct mapping from states to actions. Instead of using the estimation of the value function as a baseline for finding an optimal policy, the policy can be explicitly represented by its own function approximator [9]. These methods are advantageous in real world scenario, as the policy representation can be chosen to fit the task at hand. Instead of using the estimation of the value function as a baseline for finding an optimal policy, the policy can be explicitly represented by its own function approximator. These methods are advantageous in real world scenario, as the policy representation can

be chosen to fit the task at hand. In other words, policy-based algorithm increases the probability for the agent to choose good actions and decrease the probability of actions that lead to end states that are not desired. Yet another reason to have policy-based algorithms as subject to further development is use of fewer tuning parameters than value-based algorithms. [10] Finally, future iterations should continue working on minimising the positive giveaway, whilst continuing to abide by the e-weighing regulations.

5.3 Final conclusion

The application of RL algorithms in a batching environment were found to be successful in the limited implementation investigated in this article. It shows promise, that the further development of the iterations, could potentially prove a valuable asset in batching algorithms, and could reduce giveaway with little to no human intervention.

Acknowledgement

The authors of this work gratefully acknowledge Grundfos for sponsoring the 7th MechMan symposium.

References

[1] A. WeighTronix, "Average weight legislation a quick reference guide." https://www.averyweigh-tronix.com/en-GB/News/Weighing-tips/General-weighing-tips-and-guides-/Average-Weight-Legislation-A-quick-reference-guide/.

[2] E. C. Directive, "Council directive 76/211/eec of 20 january 1976 on the approximation of the laws of the member states relating to the making-up by weight or by volume of certain prepackaged products." https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:31976L0211&from=GA, 01 1976. (undefined 23/5/2019 7:32).

[3] T. Kvisgaard and J. Bomholt, "Method and apparatus for weight controlled portioning of articles having non-uniform weight," 03 1996.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529–533, 02 2015.

[5] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. D. JesÃºs, Neural Network Design. 2nd ed., 2014.

[6] G. E. H. Alex Krizhevsky, Ilya Sutskever, "Imagenet classification with deep convolutional neural networks,"

[7] J. Brownlee, "How to choose loss functions."

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization,"

[9] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation." https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-pdf, 09 2014.

[10] J. Peters, "Policy gradient methods scholarpedia." http://www.scholarpedia.org/article/Policy_gradient_methods, 11 2010.