# TODO Title
## TODO SubTitle

## 1  Robo Cup Logistic League Motivation

The Robo Cup Logistics League (RCLL) is designed to represent a common scenario in Industry 4.0. Here one needs to assemble products based on dynamic custom orders in a real-time environment. This means the environment is driven by a consistent stream of orders and individual delivery windows. The challenge here consists in scheduling the assembly process of orders in a feasible manner, to abide the delivery window of as many orders are possible, figure out priorities or complexities and to understand which orders can't be completed in time.

## 2  Reinforcement Learning approach

We present a solution using Reinforcement Learning (RL) to tackle the problem of deciding which assembly step the robots need to process next. Due to the complexity of the problem a Deep Q-Learning (DQL) approach is initially applied, while keeping open options and alternatives described in section 2.5.

The goal of Q-learning is for an agent to select actions, which maximize the total reward earned over an episode (=game), by observing the state and reward from an environment (see figure 1). So from a state $s$ and action $a$ we want to get the maximum future reward $r$, which we denote as the Q-value $Q(s, a) = r(s, a) + \gamma max_a Q(s', a)$. With $\gamma$ we can control the weight of the future rewards.

When shifting from Q-learning to DQL we apply a Deep Neural Network (DNN or DQN) to learn the Q-values for each individual action, given a state (see figure 2). We can train the DNN gradient with backpropagation from a squared error loss on the target $Q_\theta(s, a)$. Since in contrast to a normal DNN scenario, here we have to update the target we are learning (like cutting the branch one sits on), we need to set-up two parallel networks like in figure 3. From figure 3 we see that we work with two networks of identical achitecture and fix the target network weights, while training the prediction network. The weights are copied over to the target network every $C$ iterations and enable us to form a more stable policy $\theta$. As an addition to this we apply experience replay, by storing a number of $(s_t, a_t, r(s_t, a_t), s_{t+1})$ tuples, which we can reuse reducing total data needed.

Putting all together we arrive at figure 4, which roughly translates to

1. Giving the current state to the DQN, which returns the Q-values for all possible actions.
2. Selection a possible action with an $\epsilon$-greedy policy, where we select a random action with probability $1 - \epsilon$ or the action with the maximum Q-value $max_a Q(s, a)$.
3. Proceed to next state taking the selected action and receiving respective reward. Also storing the transition in the replay memory.
4. Sample random batches of transitions from the replay buffer to compute the loss $L = (r + \gamma max_{a'} Q_{\theta'}(s', a') - Q_\theta(s, a))^2$.
5. Perform gradient descent minimizing the loss for the prediction network.
6. Every $C$ iteration copy the weights of the prediction network to the target network.
7. Repeat all of the above for a number of episodes.

### 2.1  Environment

In the first steps we apply a more simplified environment, which only provides an order schedule with delivery windows. The initial goal is thus for the system to learn which orders are most feasible to complete or award most rewards.

We build the environment so that we can have a precision of seconds, while not specifically modelling each second as a discrete time step. The idea is to merge the discrete time into the actual state as a numerical feature, which is incremented in the expected passed time for a processing step. Initially the elapsed time for the intermediate steps is drawn from Gaussian distributions, which can be later extended with application
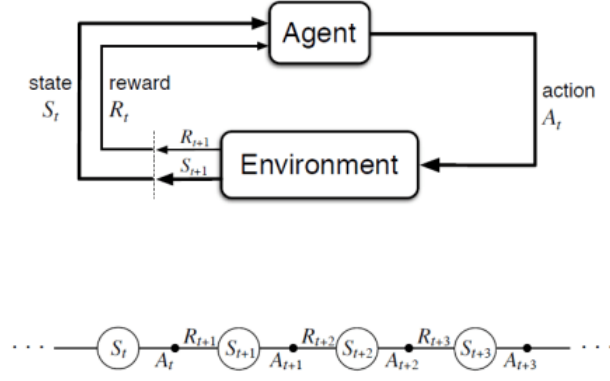
Fig. 1. Basic idea of Reinforcement Learning.
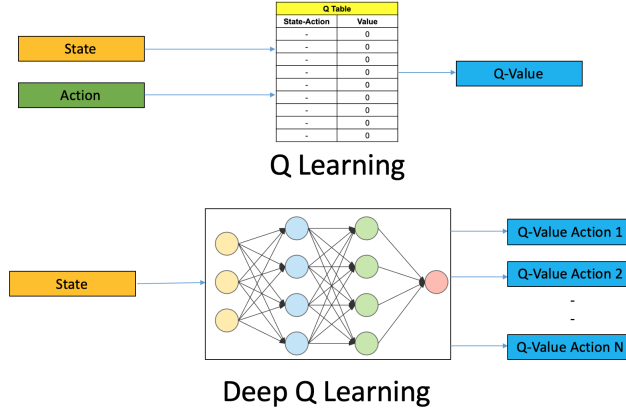


Fig. 2. Comparison of using a Q-table and a neural network.
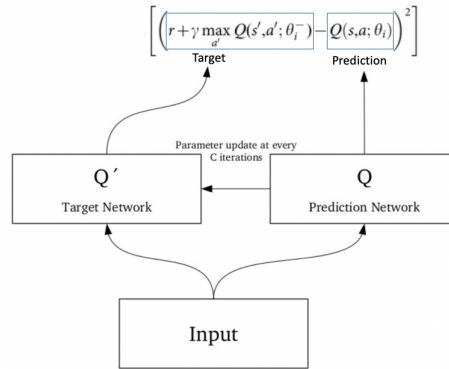


Fig. 3. Structure for updating the two internal networks in the RL set-up.

of the distance matrix and real-world data.

Each task $t_k$ will depend on the duration for a sub-step needed to complete order $O = \sum_{i=0}^{N} t_i$ and will be modelled as $t_k = \mathcal{N}(\mu_k, \sigma_k^2) + \mathcal{U}(a_M, b_M)$. Here the normal distribution $\mathcal{N}$ is given by $\mu_k = d_{ij}$ drawn from a distance matrix representing the distance between machines $M_i$ and $M_j$. The $\sigma_k$ can either be fine-tuned by hand or matched from real recorded robot data. So in other words $\mathcal{N}$ estimates the time a robot will take to
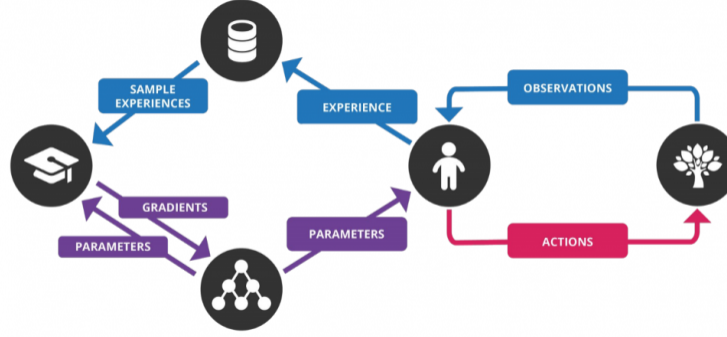
**Fig. 4.** Structure for the full DQL cycle.

move. For the uniform distribution $\mathcal{U}$ we adapt the values given from the rulebook, which represent random uniform processing time at the specific machine $M$.

The environment will handle the internal states, which include the playing field distances, product schedule, and robot locations. It will support OpenAI-gym conform functions and more optimizations. The environment can handle communication with the RefBox for the RCLL, to pull out the game field and intermediate points as rewards. The reason for this is to learn the specific official implementation and maximize the rewards based on the official tool, as there may be inconsistencies to the rulebook and the environment needs no updates on rule changes (as those are implemented in the RefBox by the official parties).

### 2.2   States

The state space consists of orders, up to 3 products in the pipeline, distance matrix of the machines, availability of machines and current time (the position of the robot should be deductible from last state). While the distance matrix is constant throughout an episode, the other parameters are simulating a dynamic real-time environment.

Specifically we can represent an order as a vector of $O = \{\mathcal{B}, \mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{C}, n, c, d_{start}, d_{end}\}$, where

- $\mathcal{B}$ represents a set of base types
- $\mathcal{R}$ is one of up to 3 ring types
- $\mathcal{C}$ is the cap type
- $n$ is the requested amount
- $c$ indicates whether the order is competitive
- $d_{start}$ is the start of the delivery window
- $d_{end}$ is the end of the delivery window

A intermediate product in the pipeline will look like $\{\mathcal{B}, \mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$, which corresponds to the physical object one of the 3 robots can transport. We can also omit the cap as we will deliver straight after.

The distance matrix is computed at the start of the episode and the machine availability is as the game dictates.

While there is still room for changes the current dimensions are ...

### 2.3   Actions

The actions consist of selecting $A = \{\mathcal{B}_P, \mathcal{R}_P, \mathcal{C}_P, d_P\}$, which is a base, ring, cap or discarding the product in pipeline $P$. By discarding we order the option to discontinue the current order and use the pipeline slot for another. In total we have an action space of $A \cdot P$.

### 2.4   Rewards

The initial idea for the rewards is that they should strongly mirror the current rewards assigned by the RefBox, to optimize the points gained in an actual game using the RefBox's rulebook implementation. There

are just minor additions to the rewards, like e.g. adding some for acquiring bases to stimulate the learning of the network. This point may see more changes later in the development process, as it is one of the main ways to regulate the learned policy.

## 2.5 Risk mitigation

For DQL to work we make sure the Markov property holds, i.e., that each state only depends on the previous state and the transition from that state to the current one. For this reason we need to make sure that the states provide all the needed information, so the above design is not completely fixed and may be updated later.
As such here one will find possible fallback strategies and properties in no particular order:

- We avoid using a multi-agent model by focusing on the order schedule and deducting a task sequence based on it. This task sequence can be forwarded from the GRIPS Teamserver to the individual robots and the time they take will be modelled as described in section 2.1.
- If we need to reduce complexity or in contrast intertwine the actual robot movement deeper into the model, we can apply similar multi-skill and multi-machine approaches as described in [QWGL16].
- In the case where the estimation of time consumption for the specific tasks do not represent the real world well enough, we can train an additional neural network in a supervised manner. Here the inputs would be from measured data for given machine distance matrix - to give us more accurate estimations for the expected time consumption at a specific task.
- When the communication with the RCLL RefereeBox, becomes problematic the rules can be manually translated into the environment.
- Initially we will model by only allowing allowing valid actions (similar to Google Deep Mind), which consist in allowing only process known orders (not guessing future orders), but this can be changed by assigning negative rewards to the guessed order steps.
- There is a possibility that the model will possibly disregard the delivery windows as (per official reward structure) the assembly of more difficult products yields much more then the timely delivery.

# 3 Main development schedule

First we try to build a network which can assemble a single order by selecting from the actions. Next we want it to recognize and understand the time it will take to process a single order, with a constant distance between machines. To do this we want it to be able to select from a number of orders and try to understand the delivery window.
The environment will be extended as needed and the reward and action space will be adjusted as well. This should enable scaling up the difficulty and observe the feasibility of the DQN approach.

## References

QWGL16. Shuhui Qu, Jie Wang, Shivani Govil, and James O Leckie. Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: An ontology-based, multi-agent reinforcement learning approach. *Procedia CIRP*, 57:55–60, 2016.