Deep Reinforcement Learning for Multi-Age Inventory Management with Complex Production Actions

Final Presentation for the Master Thesis

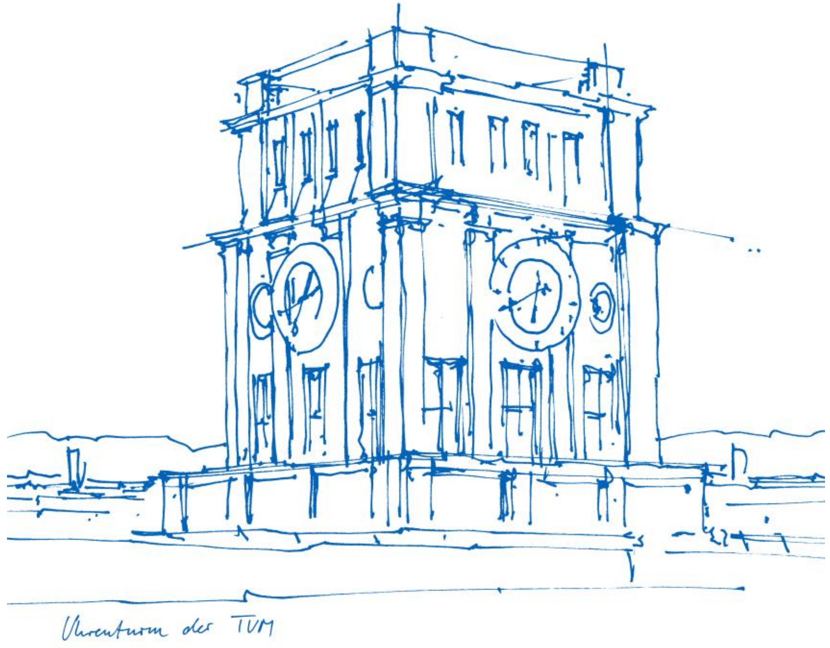
Student: Thu Ha Dao (03721313)

Supervisor: Alexander Pahr

Prof. Martin Grunow

Chair of Production and Supply Chain Management

11th March 2022



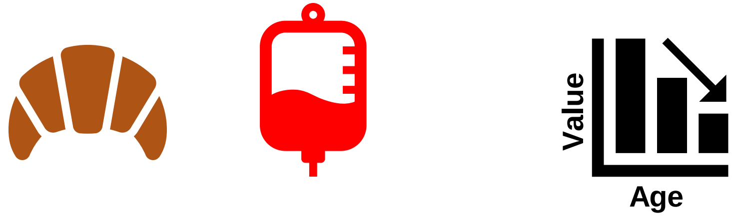
Agenda

Agenda

Inventory management of multi-age products

Multi-age products

- Products controlled based on ages
- Value/ Quality varied with ages
  - ➢ Value decreases over time: perishable food products, blood platelets
  - ➢ Value increases over time: forest, wine



Complex production action

- Due to multi-age characteristic (differentiated demand for each age class)
- Due to flexibility in issuing / production action: blending (wine), substitution (blood platelet, perishable food products)

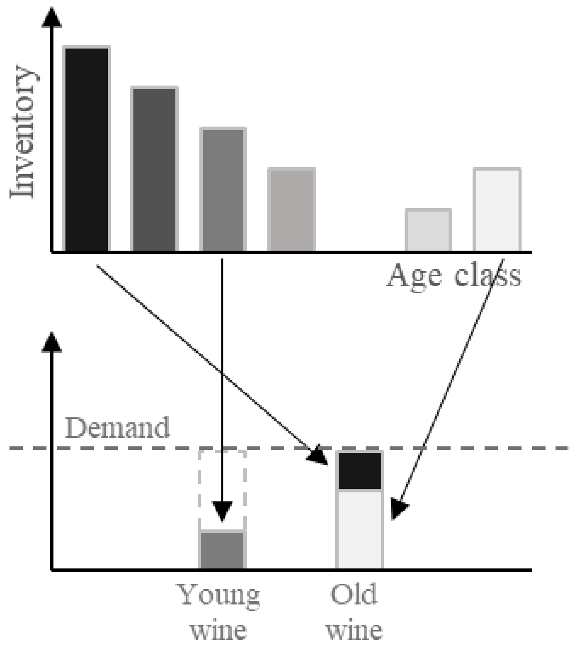
Port wine inventory system is an example of complex production action

Branded port wine products

- Value increases with storage time (ages)
- Age indication (i.e., *target age*) = minimum average storage time of the wine inside

Blending decision

- Mix wines from different age classes together to achieve a blend satisfying target age



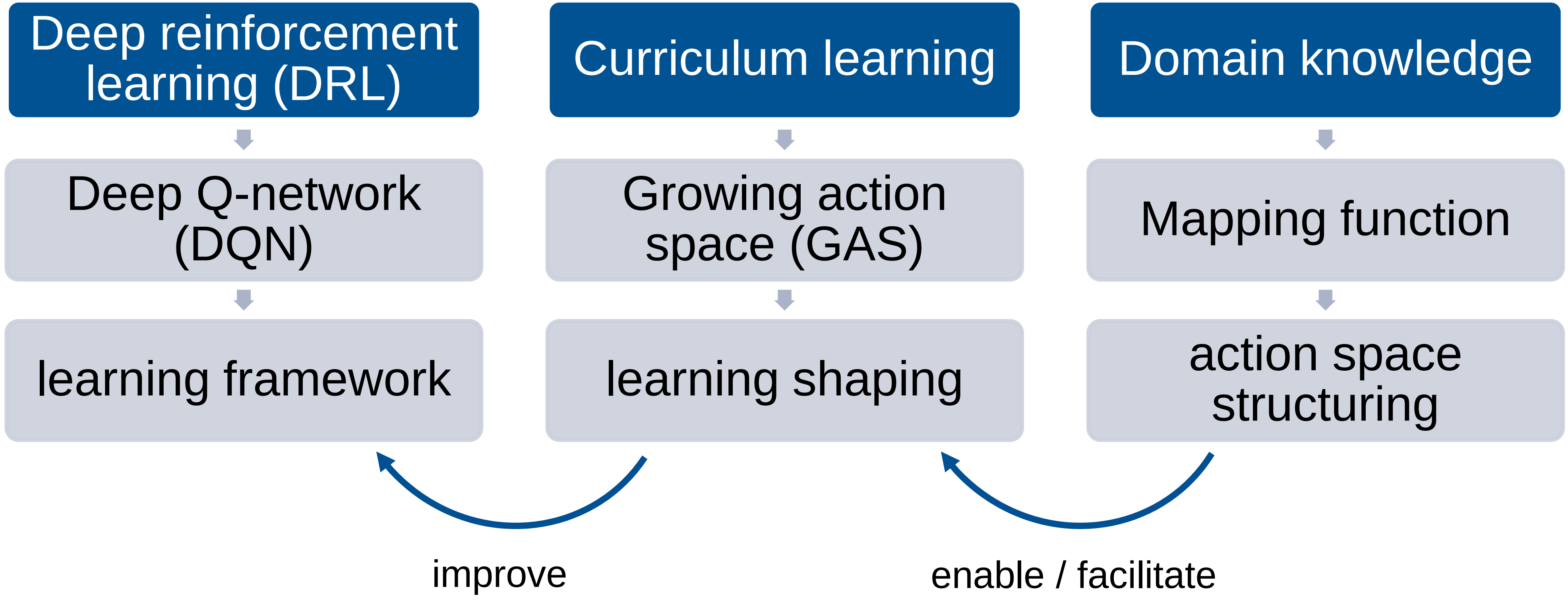
Trade-off:
flexibility
vs.
complexity

Final production action as a *sequence* of decisions:

i. *How many* demand units will be satisfied for each product?

ii. *How* to blend those demand units?

Main components



Research questions

Research questions

[RQ1]: How can we use domain knowledge/ understanding about problem to improve the performance of DRL in the context of a multi-age inventory management problem with complex production action?

[RQ2]: How to apply GAS-based curriculum learning into a combinatorial action space with sequential sub-actions?

[RQ3]: How is the robustness of the proposed methodology, especially in terms of the choice of mapping functions?

Agenda

Multi-age inventory management with complex actions

Paper	Optimized decision	Method	Rule-based solution	Action complexity		
				AD	S	CS
Nahmias and Pierskalla (1976)	Ordering	A	No		✓	
Cohen et al. (1981)	Ordering	A	Yes			✓
Goh et al. (1993)	Issuing	A	Yes	✓	✓	
Deniz et al. (2010)	Ordering + Issuing (pair)	A	Yes	✓	✓	
Deniz et al. (2020)	Ordering + Issuing (pair)	A	Yes	✓	✓	
Haijema et al. (2007)	Ordering	N	No	✓	✓	
Civelek et al. (2015)	Ordering + Issuing	N	Yes	✓	✓	✓
Chen et al. (2021)	Ordering + Issuing	A, N	No	✓	✓	✓
Abbaspour et al. (2021)	Ordering	N	Yes	✓	✓	
Pahr et al. (2021)	Ordering + Issuing	N	No	✓	✓	✓
Hendrix et al. (2019)	Ordering	N	Yes		✓	
This thesis	**Issuing**	**N**	**No**	✓	✓	✓

R – Rule-based setup, A – Analytical analysis, N – Numerical analysis

AD – Age-differentiated demands, S – Substitution, CS – Combination of sub-actions

There is a recent surge in research of application of DRL into inventory management problem (Boute et. al., 2021)

Paper	DRL algorithm	Decision	Inventory setting	Transfer learning between / from
Gijsbrechts et al. (2021)	A3C	Replenishment + distribution	ME	
Oroojlooyjadid et al. (2021)	DQN	Independent ordering with cooperative goal	ME	Agents
Harsha et al. (2021)	PARL	Replenishment + distribution	ME	
Sultana et al. (2020)	A2C	Replenishment + distribution	MP, ME	Products
Meisheri et al. (2021)	DQN, PPO	Replenishment + distribution	MP, ME	Products
Moor et al. (2021)	DQN	Ordering	MA	Heuristics
This thesis	**DQN**	**Production (Sales + Blending)**	**MA**	**Domain knowledge**

ME – Multi-echelon, MP – Multi-product, MA – Multi-age

DRL with (large) discrete combinatorial action space

Paper	Methodology	Remarks
ANN configuration		
Zhao et al. (2018)	Adjusting components of DRL	Changing input and output of ANNs in DQN algorithm
Zahavy et al. (2018)	Adding supplemental modules	A separate ANN to eliminate irrelevant actions
van de Wiele et al. (2020)		A separate ANN to predict the best-known action
You et al. (2020)		ANN module to learn values of state decompositions
Delarue et al. (2020)	Partially replacing ANNs	DRL integrated with mix-integer programming
Harsha et al. (2021)		DRL integrated with mix-integer programming
Cappart et al. (2021)		DRL integrated with constraint programming

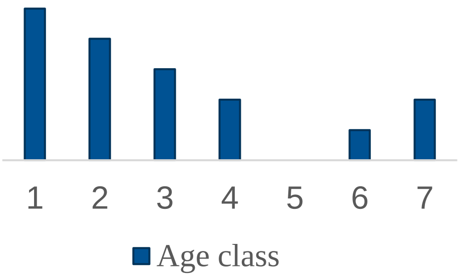
Paper	Methodology	Remarks
Action space transformation		
Dulac-Arnold et al. (2016)	Embedding actions	Discrete action embedded into continuous one
Kanervisto et al. (2020)	Removing actions Combining actions	Review of DRL algorithms applied in video game
Farquhar et al. (2020)	**Structuring action spaces**	Action space divided into restricted sub-spaces
Bamford and Ovalle (2021)		Action space decomposed into a sequence of sub-spaces
This thesis		**GAS extended to action space with sequential sub-tasks**

Agenda

Problem setting

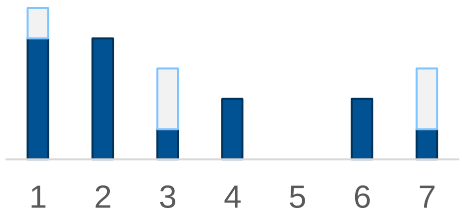
- Simplified from the setting from Pahr et al. (2021)
- Only consider the blending state with decay risk, without blending state and price uncertainty

(Current) State



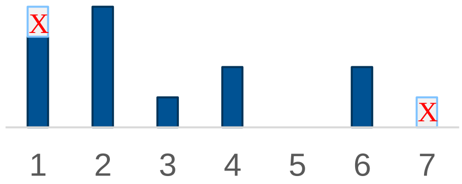
① **Sales + Blending action**

	Young	Old
Sales (qty)	2	3
Blending (age)	3, 3	1, 7, 7



② **Outdate & Decay**

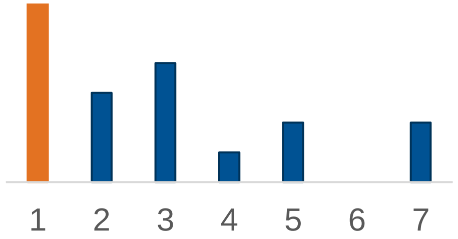
- Last-age stock outdated
- Random stock decayed (e.g., 1 unit age class 1)



③ **Age & Purchasing**

- All stocks aging by 1 year
- Base-stock ordering policy
- New stock added to first age class

(Next) State



Problem formulation

State

S inventory states, $s = (s_1, \dots, s_{i^{max}}) \in S$, $|\boldsymbol{S}| = (\boldsymbol{inv^{max}} + \mathbf{1})^{\boldsymbol{i^{max}}}$

s_i inventory level in age class i of state s

Parameters & Sets

I age classes, $i \in I = \{1, \dots, i^{max}\}$

B brand products, $b \in B = \{1, \dots, b^{max}\}$ with target ages i_b^{trg}

Inv discrete inventory levels, $inv \in Inv = \{0, \dots, inv^{max}\}$

d_b deterministic/stationary demand for brand product $b \in B$

cb_b profit contribution of brand product b

cs_i supermarket profit contribution for age class i

hc holding costs

pp purchase price

Action

A **production actions**, $a = (sl_a, bl_{(sl_a)}) \in A$

SL **sales sub-actions**, $sl = (f_{1,sl}, \dots, f_{b^{max},sl}) \in SL$, $|\boldsymbol{SL}| = \prod_{b \in B} d_b$

$f_{b,sl}$ demand fulfillment for product b in sl

BL_{sl} **blending sub-actions** corresponding to sales sub-action sl, $bl = (h_{1,bl}, \dots, h_{i^{max},bl}) \in BL_{sl}$

$h_{i,bl}$ quantity of stock age class i used in bl

State transition probability

$p_{i,s\,s'}^{decay}$ probability that stock of age class i changes from s_i to s_i' due to decay, 0 if $s_i' > s_i$ and $p_{i,s^{pre}(s,a),\,s'}^{decay} = p_{i,\,s^{pre}(s,a),\,s'}^{rD} \cdot ol$ otherwise

$p_{i,s,s'}^{rD}$ relative risk of decay following truncated discrete Weibull distr.,

ol overall probability that an inventory unit decays during its lifetime

$s^{pre}(s,a)$ (deterministic) pre-decay state obtained by taking a on state s

$$p(s'|s,a) = \prod_{i \in I} p_{i,s^{pre}(s,a),\,s'}^{decay}$$

Problem formulation

Reward function

Expected reward by taking action a in state s:

$$r(s,a) = r\left(s, (sl_a, bl_{sl_a})\right)$$

$$= \sum_{b \in B} cb_b \cdot f_{b,sl_a} \quad \text{profit from selling branded products}$$

$$+ \underbrace{cs_{i^{max}} \cdot o_{s,a}}_{\text{profit from selling outdated units}} + \sum_{s'} \left(p(s'|s,a) \cdot \underbrace{\sum_{i \in I} cs_i \cdot (s_i^{pre}(s,a) - s_i')}_{\text{profit from selling decayed units}} \right)$$

$$- hc \cdot \left(\underbrace{\sum_{i \in I} i \cdot h_{i,bl_{sl_a}} - \sum_{b \in B} i_b^{trg} \cdot f_{b,sl_a}}_{\text{additional holding cost}} \right) - \underbrace{\sum_{s'} p(s'|s,a) \cdot pp \cdot g_{s'}}_{\text{purchasing cost}}$$

with cb_b = profit contribution of brand product b

cs_i = supermarket profit contribution per unit of age class i

hc = holding costs

pp = purchasing price

$g_{s'}$ = purchased quantity based on order-up-to level BS

$o_{s,a}$ = outdating quantity for state-action pair s, a

Value functions

- The objective is to find a policy π^* maximizing the discounted expected return
- Following the instruction in Sutton & Barto (2018), we define the state-value function for policy π in state s as :

$$V_\pi(s) = r(s,a) + \sum_{s'} p(s'|s,a) \cdot \gamma \cdot V_\pi(s'')$$

with γ = discount rate, $0 < \gamma < 1$,

s' = from state s after taking action a and realizing decay risk

s'' = (deterministic) state obtained from s' after purchasing

and the Bellman optimality equation for optimal value function V^* as :

$$V^*(s) = \max_a \left\{ r(s,a) + \sum_{s'} p(s'|s,a) \cdot \gamma \cdot V^*(s'') \right\}$$

- The optimal policy π^* can be found by applying value iteration algorithm, as described in Sutton & Barto (2018), with termination condition as described in Odoni (1969)

Agenda

Reinforcement Learning & Q - learning

Considering a MDP problem with the Bellman optimality equation for the optimal action value function as follows :

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right]$$

$$= \sum_{s'} p\,(s'|s,a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$$

- **Temporal-difference learning**: combination of Monte Carlo methods (sampling from raw experience) and dynamic programming (bootstrapping on current estimate). The simplest TD update rule has the form of:

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma\, V(S_{t+1})}_{\text{target for update}} - V(S_t)]$$

- **Q-learning**: an *off-policy temporal difference control* algorithm presented in Watkins (1989), approximating the optimal action value function independently with the policy being followed. The update rule is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \underbrace{\left[R_{t+1} + \gamma \underbrace{\max_{a} Q(S_{t+1}, a)}_{\text{estimation of } V(S_{t+1})} - Q(S_t, A_t)\right]}_{\text{Temporal difference used for policy evaluation}}$$

(Sutton & Barto, 2018)

Deep Reinforcement Learning and Deep Q-learning

- In their work, Mnih et al. (2015) presented **Deep Q-network (DQN),** a novel method combining deep neural networks and Q-learning to improve the stability when applying a nonlinear function approximator (i.e., neural network) to estimate action value function.

$$\boldsymbol{Q}(\boldsymbol{s},\boldsymbol{a};\boldsymbol{\theta}) \approx Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^{*}(s',a') \,|\, s,a\right]$$

nonlinear function approximator

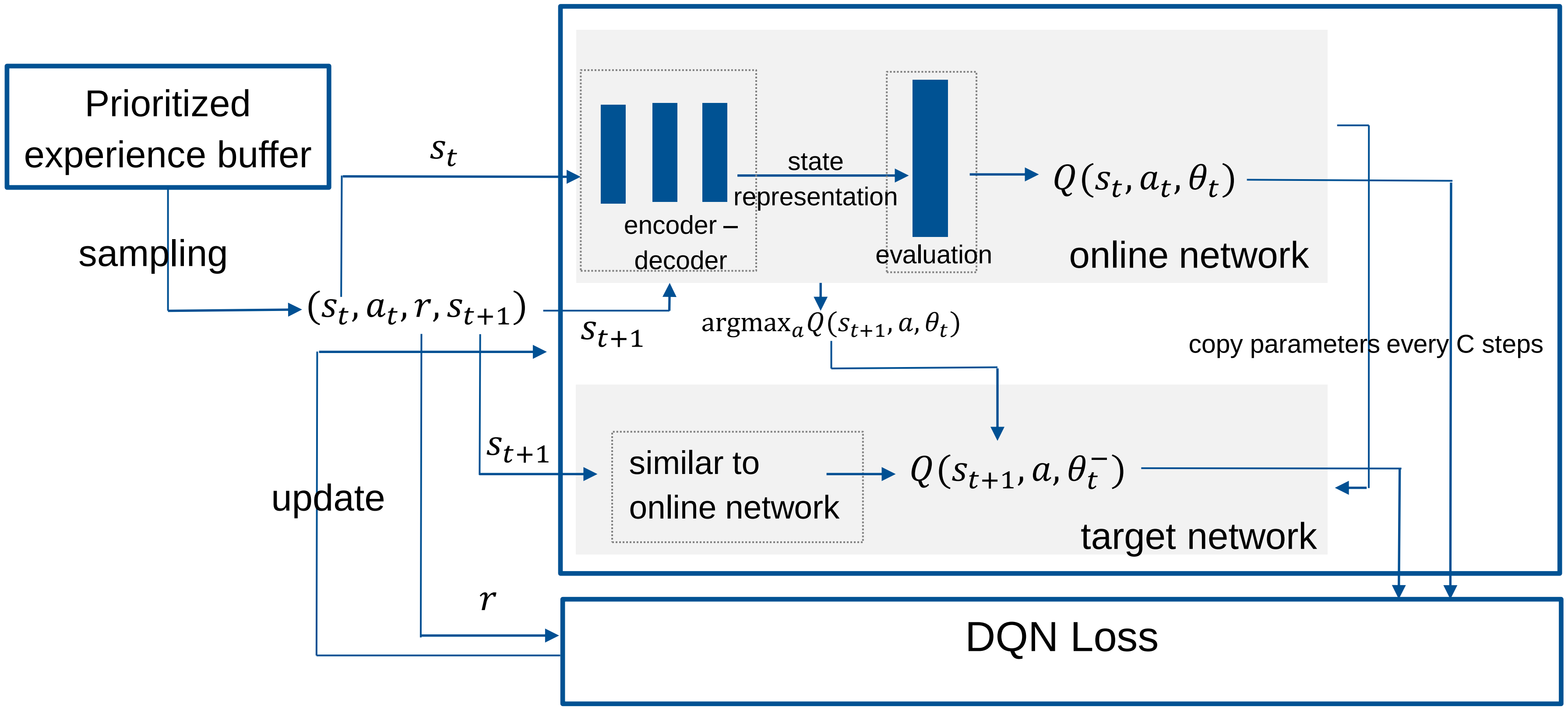
- Two main ideas in DQN:
  1. **Experience replay**: Q-learning update using samples (or minibatches) of experience from the pool of stored samples
  2. **Target network**: a separate network for generating the targets in the Q-learning update. It is similar to online network

and updated every C steps by copying parameters from online network

$$\theta_{t+1} \leftarrow \theta_t + \alpha\left[R_{t+1} + \underbrace{\gamma \max_{a} Q(S_{t+1}, a; \boldsymbol{\theta}_{\boldsymbol{t}}^{-})}_{\text{target network}} - \underbrace{Q(S_t, A_t; \boldsymbol{\theta}_{\boldsymbol{t}})}_{\text{online network}}\right] \nabla_{\theta_t} Q(S_t, A_t; \boldsymbol{\theta}_{\boldsymbol{t}})$$

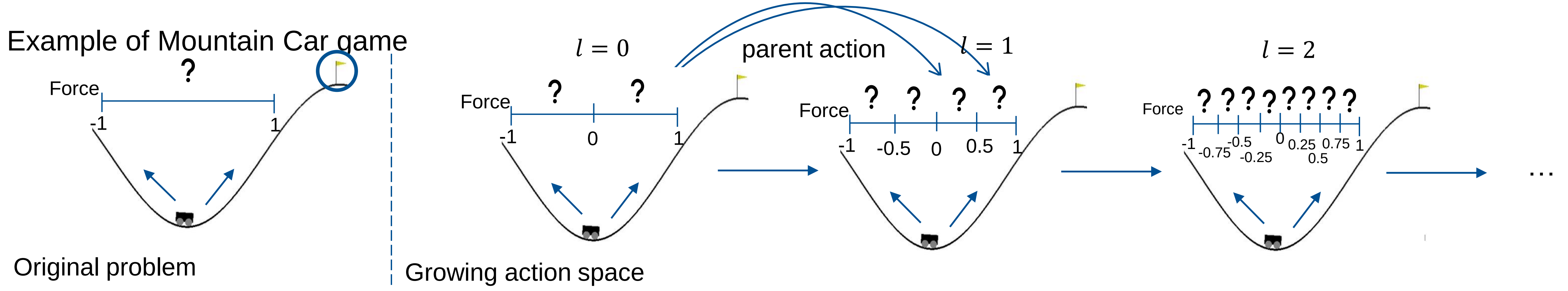
- Some improved variants of DQN
  - *Double DQN* (Hasselt et al., 2016): transforming the max operator in the target into action selection and action evaluation to prevent over-optimistic
  - *Prioritized experience replay* (Schaul et al., 2016): sampling strategy to increase learning efficiency

DQN with Double DQN and Prioritized DQN



Curriculum learning with Growing action space

- Farquhar et al. (2020) introduced **Growing action space (GAS)**, a curriculum learning paradigm to take advantage of the hierarchies in action space to accelerate Q-learning
- Main ideas:
  - ➢ start training agent with a restricted action space to intentionally guide the agent toward meaningful experiences
  - ➢ gradually expanding the action space to the full space to find the optimal policy
- Assumption on the action space:
  - ➢ **Hierarchies** in the action space:
    - ▪ $more\ restricted\ action\ space\ \subset\ less\ restricted\ action\ space$
    - ▪ identifying for every action $a \in A_l, l > 0$, a *parent action* $parent_l(a)$ in the space of A_{l-1}
  - ➢ **Learning in restricted action space is more meaningful** than randomly exploration in the whole action space
  - ➢ Policies are **easier to learn in restricted action spaces** than in the full space



Curriculum learning with Growing action space (cont.)

Formulation

- Action space A is divided into N action spaces A_l, with $l \in \{0, \dots, N-1\}$, satisfying the hierarchical condition:
$$A_0 \subset A_1 \subset \dots \subset A_{N-1} \subseteq A$$
- Optimal policy $\pi_l(a|s)$, optimal action value function $Q_l^*(s|a)$ and optimal state value function $V_l^*(s) = \max_a Q_l^*(s, a)$ corresponding to restricted action space A_l
- **Value estimation:** $V_i^*(s) \leq V_j^*(s)\ \forall s\ \ if\ i < j\ with\ i, j \in \{0, \dots, N-1\}$ as $A_i \subset A_j$
$$\hat{Q}_{l+1}^*(s, a) = \hat{Q}_l^*(s, parent_l(a)) + \triangle_l\ (s, a)$$
- Modified Bellman optimality equation:
$$Q_l^*(s, a) = \mathbb{E}\left[r(s, a) + \gamma \max_{i \leq l} \max_{a'} Q_i^*(s', a')\right]$$
Q-functions at boarder action space to be bootstrapped from those at more restricted one
- Other characteristics:
  - ➢ Off-action-space learning: using experience with action in A_l to update all estimation of optimal value function corresponding to “higher” action space $\hat{Q}_{\geq l}^*(s, a)$
  - ➢ Model-free and off-policy algorithm

Integration of domain knowledge into DQN through GAS

- Based on the original GAS, we formularize an approach to integrate domain knowledge to DQN.
- **Assumption**:
  - consider an action **composing of sub-actions**, i.e.,

    $$A = A_0^{sub} \times A_1^{sub} \times \ldots \times A_{N-1}^{sub} \text{ and } a = (a_0^{sub}, a_1^{sub}, \ldots, a_{N-1}^{sub}) \text{ with } a \in A, a_l^{sub} \in A_l^{sub}, l \in \{0, \ldots, N-1\}$$

  - sub-actions are **interdependent** and could be implemented **in order**: $a_0^{sub} \to a_1^{sub} \to a_2^{sub} \to \ldots \to a_{N-1}^{sub}$ and $seq_l = (a_0^{sub}, a_1^{sub}, \ldots, a_l^{sub})$ (i.e., sequence of sub actions until l)
  - if seq_l is defined, **domain knowledge** could be used to find (not necessarily optimal) the **next sub-action** a_{l+1}^{sub}, i.e., $\exists\ map: (seq_l, \cdot) \longmapsto a_{l+1}^{sub}\ \forall\ l \in \{0, \ldots, N-2\}$ (with $\cdot$ denoting additional parameters)
- **Definition of restricted action space:**

  ´Define each restricted action space A_l , so that action $a_l \in A_l$ is defined as:

  $$a_l = \Big(\overbrace{a_0^{sub}, \ldots, a_l^{sub}}^{\text{defined}}, \overbrace{map_{(1)}\big(seq_l^{a_l}, \cdot\big), \ldots, map_{(N-1-l)}\big(seq_l^{a_l}, \cdot\big)}^{\text{derived}}\Big)$$

  with $seq_l^{a_l}$ being the sequence of sub-actions until l followed in action a_l and $map_{(n+1)}(\cdot) = map\left(map_{(n)}(\cdot)\right)\ for\ n \geq 1$

  Therefore, we have: $A_0 \subset A_1 \subset \ldots \subset A_{N-1} \subseteq A$ and in action space A_{l-1}, parent action of a_l is:

  $$parent_{l-1}(a_l) = \left(a_0^{sub}, \ldots, a_{l-1}^{sub}, map_1\big(seq_{l-1}^{a_l}, \cdot\big), \ldots, map_{N-l}\big(seq_{l-1}^{a_l}, \cdot\big)\right)$$

Integration of domain knowledge into DQN through GAS (cont.)

Example in the multi-age setting:

- The action in the original problem is divided into 3 sub-actions:

  full action = (sales qty of b_1, sales qty of b_2, blending action) or $a = \left(f_1, f_2, bl_{(f_1,f_2)}\right)$,
- The mapping function also takes the state (inventory) as an input. To simplify, we consider the case when the inventory is full for all age classes ➔ all actions are valid.
- In this case, we define the mapping function: (1) select the highest sales qty for b_2, (2) use stock from target ages to blend.

It means $map(seq_l)$: $\begin{cases} seq_0 = (f_1) \longmapsto 2 \\ seq_1 = (f_{1,}, f_2) \longmapsto (0,0, f_{1,} 0, f_{2,} 0,0) \end{cases}$

in case of 2 branded products b_1, b_2 , target ages $i_{b_1}^{trg}, i_{b_2}^{trg} = 3,\ 5$, demands $d_1, d_2 = 2, 2$, highest inventory level $i^{max} = 6$

A_0 (Sales b_1) — seq_0 / a_0:

- 0: $\mathbf{0}, \mathbf{2}, (0,0, \mathbf{0}, 0, \mathbf{2}, 0,0)$
- 1: $\mathbf{1}, \mathbf{2}, (0,0, \mathbf{1}, 0, \mathbf{2}, 0,0)$
- 2: $\mathbf{2}, \mathbf{2}, (0,0, \mathbf{2}, 0, \mathbf{2}, 0,0)$

parent parent parent

A_1 (Sales) — seq_1 / a_1:

- 1,0: $\mathbf{1}, \mathbf{0}, (0,0, \mathbf{1}, 0, \mathbf{0}, 0,0)$
- 1,1: $\mathbf{1}, \mathbf{1}, (0,0, \mathbf{1}, 0, \mathbf{1}, 0,0)$
- 1,2: $\mathbf{1}, \mathbf{2}, (0,0, \mathbf{1}, 0, \mathbf{2}, 0,0)$

parent

A_2 Sales & Blend — seq_2 / a_2:

- $\mathbf{1}, \mathbf{1}, (0,0, \mathbf{1}, 0, \mathbf{1}, 0,0)$
- $\mathbf{1}, \mathbf{1}, (0,0, \mathbf{1}, 0,0, \mathbf{1}, 0)$
- $\mathbf{1}, \mathbf{1}, (0,0, \mathbf{1}, 0,0,0, \mathbf{1})$
- …

Global exploration vs. Hierarchical exploration

- **Global exploration** = on the scale of the whole action (sub-)space currently considered
- **Hierarchical exploration** = restricted to only a part of the current action sub-space of which all actions have the same parent action as the greedy action

  ➔ *learned knowledge of the sub-actions at lower levels (i.e., parent action) is still taken into account*

Algorithm: ϵ-greedy action selection

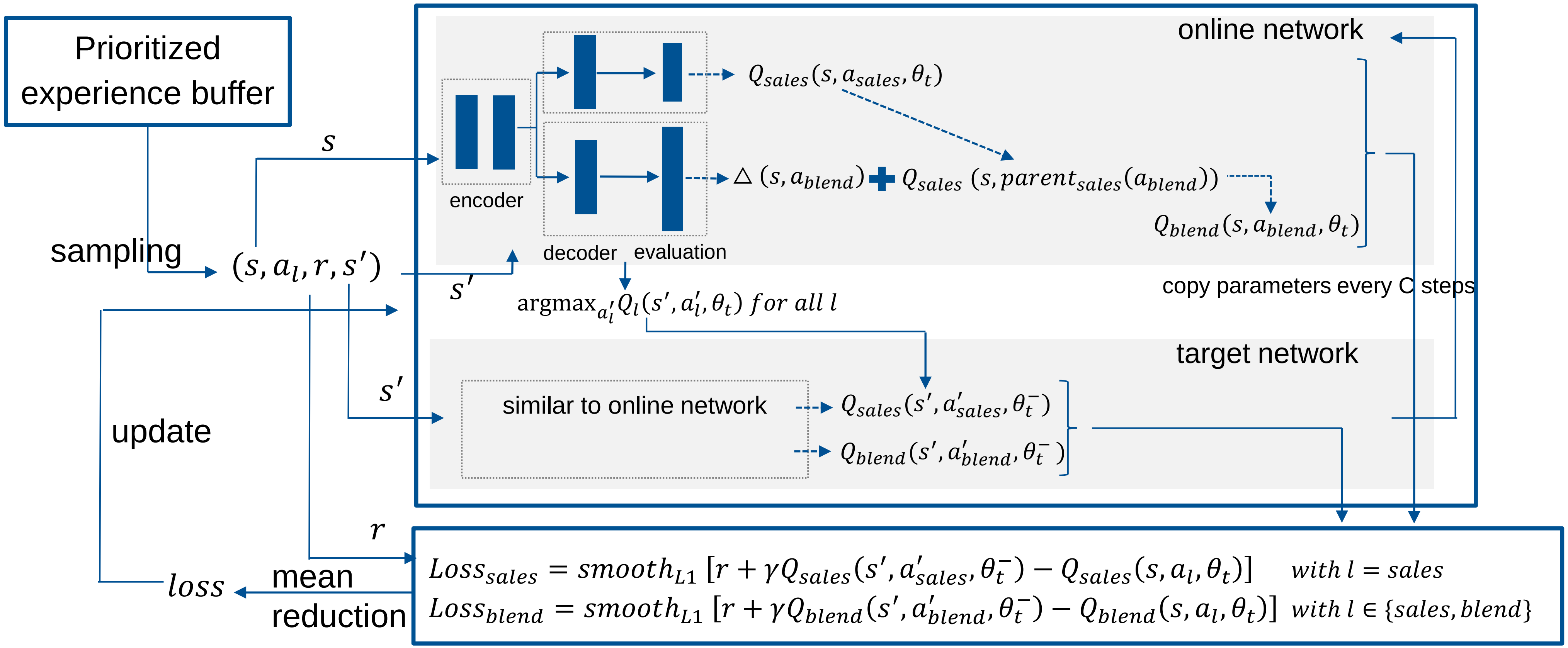
```
initialize ϵ;
p ← random();
if p ≥ ϵ then
    action ← arg max_a Q(a);
else
    action ← a random action;
end
return action;
```

Algorithm: Double ϵ-greedy action selection in action sub-space A_l

```
initialize ϵ, ϵ^h;
if l = 0 then
    perform ϵ-greedy; // double ϵ-greedy is not applicable in A_0
else
    greedy_action ← arg max_{a∈A_l} Q(a);
    p ← random();
    if p ≥ ϵ then
        // fully exploitation
        action ← greedy_action
    else
        // exploration
        p^h ← random();
        if p^h ≥ ϵ^h then
            // hierarchical exploration
            A_l^h ← set of actions a^h : a^h ∈ A_l, parent_{l−1}(a^h) =
             parent_{l−1}(greedy_action);
            action ← a random action in A_l^h;
        else
            // global exploration
            action ← a random action in A_l;
        end
    end
    return action;
end
```

We apply GAS with $N = 2$ into the case study

- In this work, we consider the case $N = 2$ (i.e., two last levels in the previous example) :
  - A_0 (restricted action space) : defined sales action with derived blending action
  - $A_1 = A$ (full action space) : defined sales action and blending action



Agenda

Value iteration – Optimal policy

Similar to case study in Pahr et al. (2021) with extended state space and action space

Case study settings:

$|I| = \mathbf{7}$

$|Inv| = 7$

$B = \{3yo, 5yo\}$

$d_b = \mathbf{3} \quad \forall b \in B$

$BS = 25$

- $|\boldsymbol{S}| =$ **823,543** (or 7^7)
- $|\boldsymbol{A}| =$ **3456**
- Transition probability table P has dimension of $(7^6, |A|, 7^6)$

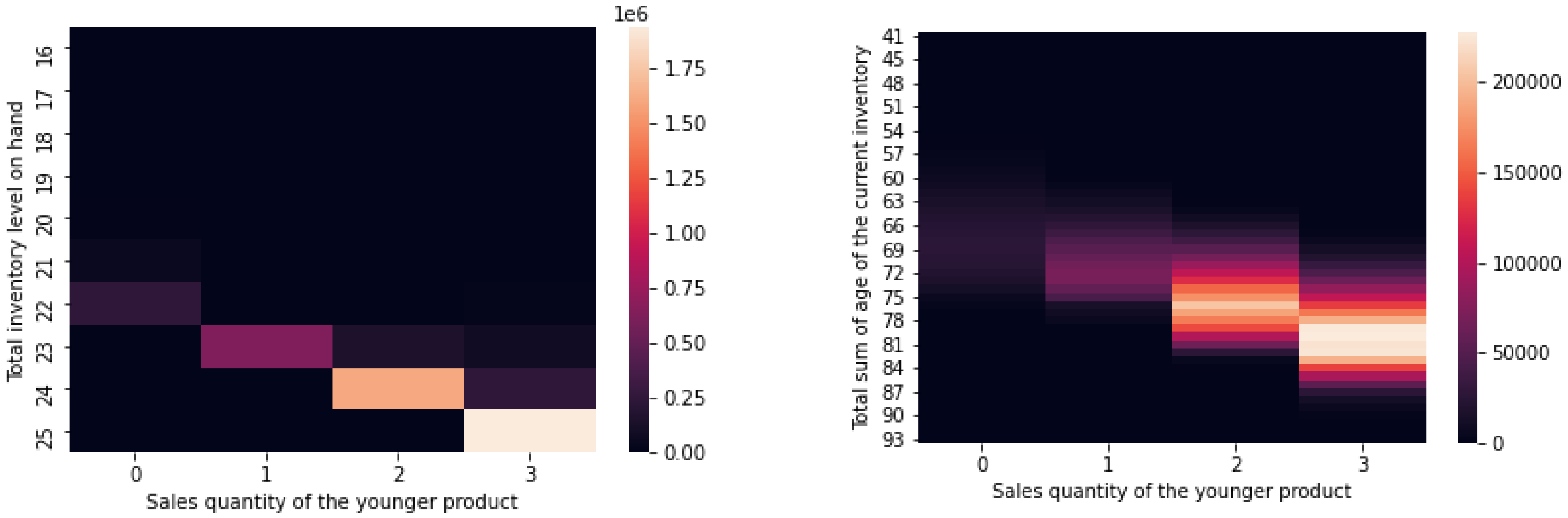
➔ $|\boldsymbol{P}| \approx \mathbf{4.8\ e^{13}}$

- Value iteration algorithm solved in 17 iterations ($\epsilon = 10^{-2}$)
- Total running time ~ 9 hours

Simulation (5M iterations)		
Visited state (% of state space)	9658 (1.2%)	
Taken action (% of action space)	111 (3.2%)	
Most frequently visited state (% of iterations)	[6, 4, 4, 4, 3, 2, 2] (1.6%) [6, 4, 4, 3, 3, 2, 2] (1.4%)	
Most frequently taken blending actions (% of iterations)	[**2**, 0, 0, 0, 1, 0, **2**] (19.1%) [**2**, 0, 1, 0, 1, 0, **2**] (10.5%)	
Average blending action	[1.55, 0.07, 0.32, 0.54, 1.04, 0.24, 1.46]	
Average inventory level	[6.0, 4.5, 4.0, 3.5, 2.8, 1.7, **1.5**]	
Average outdating	**0.0**	
Simulation results	**3yo wine**	**5yo wine**
Average sales (in units)	**2.2**	**3.0**
Underfulfillment (% of iterations)	**54.00%**	**0.09%**
Average reward / CoV	**2831 / 10.5%**	

Value iteration – Optimal policy

Positive correlation between the sales quantity of the younger product and the current total inventory level before making production decision as well as total sum of age of the current inventory



DQN & GAS-DQN training

Training hyperparameters

- Hyperparameter tunning: manual search, based on the values from the original GAS paper
- Repeat training 5 runs for each model
- Each episode includes 1000 consecutive periods, repeated 100 times per run

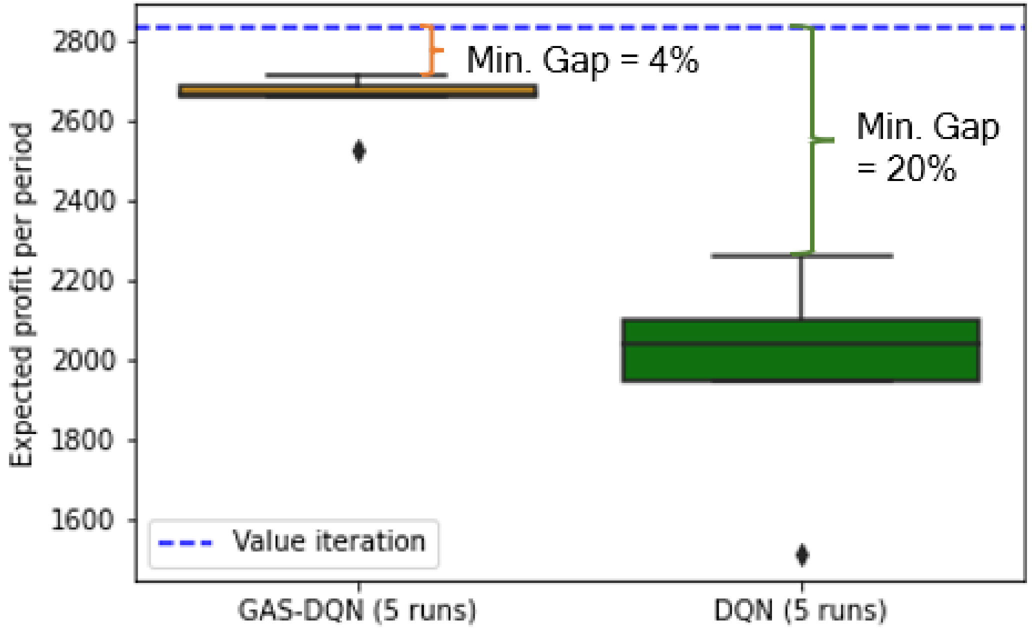
Mapping function:

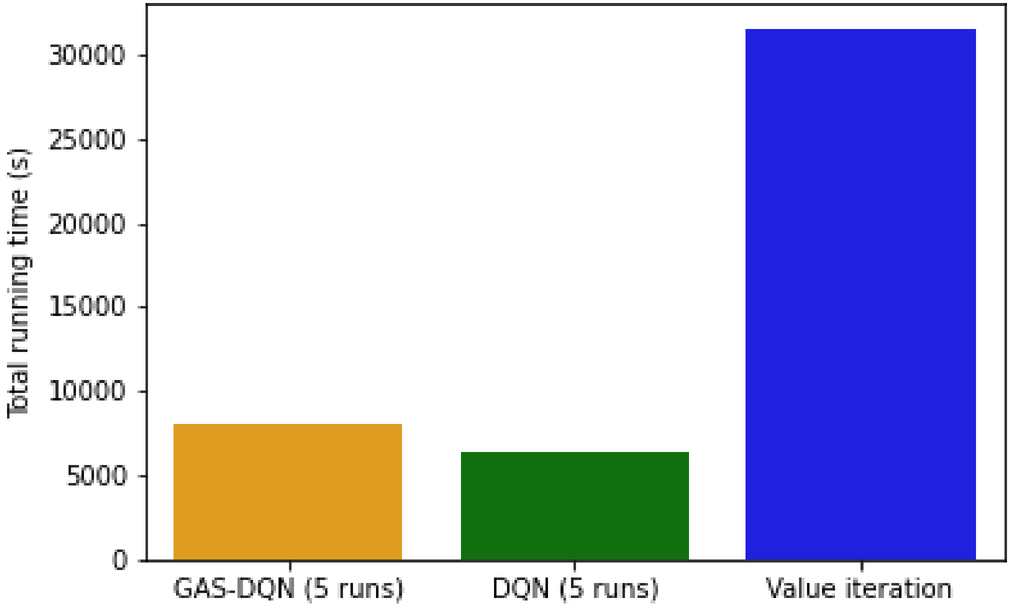
Based on the sales quantity and current inventory, select the valid blending action based on priority order as follows:

1. Selected action has the highest possible number of inventory units from the last age class, i.e., $h_{i^{max}, bl_{sl}} = \min\left(s_{i^{max}}, f_{1,sl} + f_{2,sl}\right)$
2. Selected action has the lowest possible sum of ages, i.e., $bl_{sl} = \underset{bl}{\operatorname{argmin}} \sum_{i \in \{1,\dots,i^{max}\}} i \cdot h_{i^{max}, bl}$ and $\sum_{i \in \{1,\dots,i^{max}\}} i \cdot h_{i^{max}, bl_{sl}} \geq \sum_{b \in \{1,2\}} i_b^{trg} \cdot f_{b,sl}$
3. Selected action has the lowest total decay risk (or in other words, the highest probability that there is no decayed unit), i.e.,

$$bl_{sl} = \underset{bl}{\operatorname{argmax}}\, p^{decay}_{i, s^{pre}(s,a),\, s^{pre}(s,a)} \text{ with } a = (sl, bl)$$

DQN & GAS – DQN vs. Value iteration



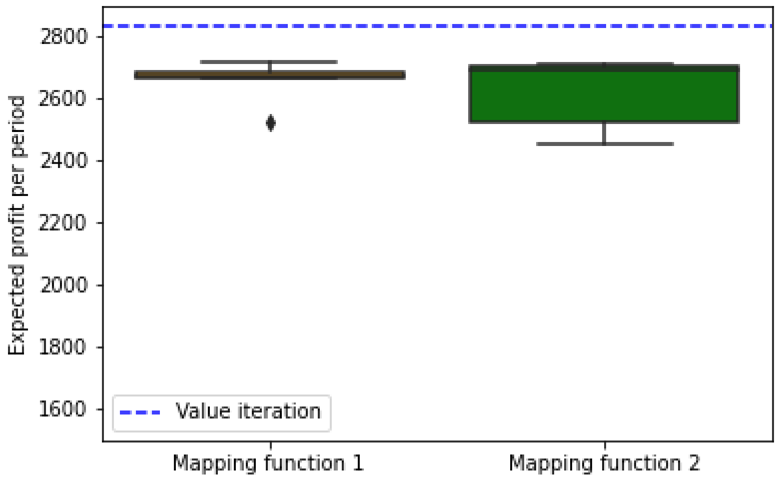


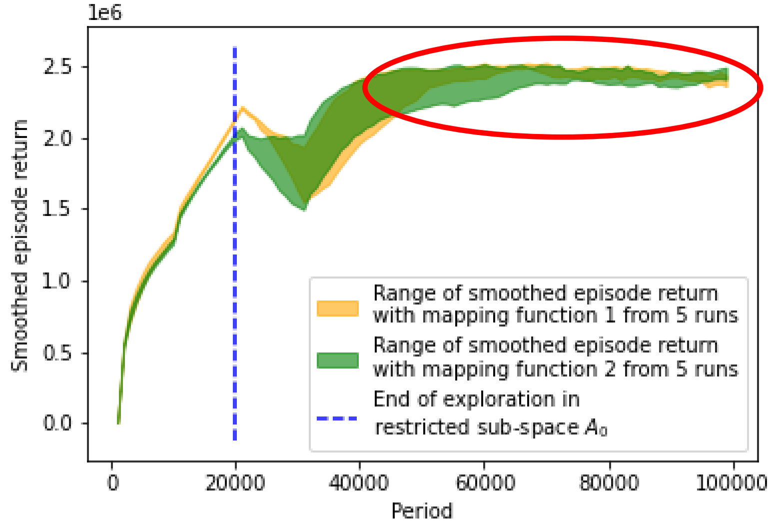
GAS – DQN results: Dive deep

	Run				
	1 (best)	2	3	4	5
Avg. total inventory level	24.5	24.9	23.2	24.0	24.3
Avg. blending spread	4.6	3.7	3.5	3.0	3.7
Avg. outdated units	0.0	0.0	0.0	0.0	0.0
Avg. sales quantity of younger branded product	1.8	1.1	2.4	2.0	2.1
Avg. sales quantity of older branded product	3.0	3.0	2.8	2.9	2.9
Avg. reward	2714	2526	2663	2687	2669
CoV of immediate rewards in one run	6.8%	7.0%	29.6%	20.5%	16.5%

Experiment 1: Mapping function (Robustness analysis)

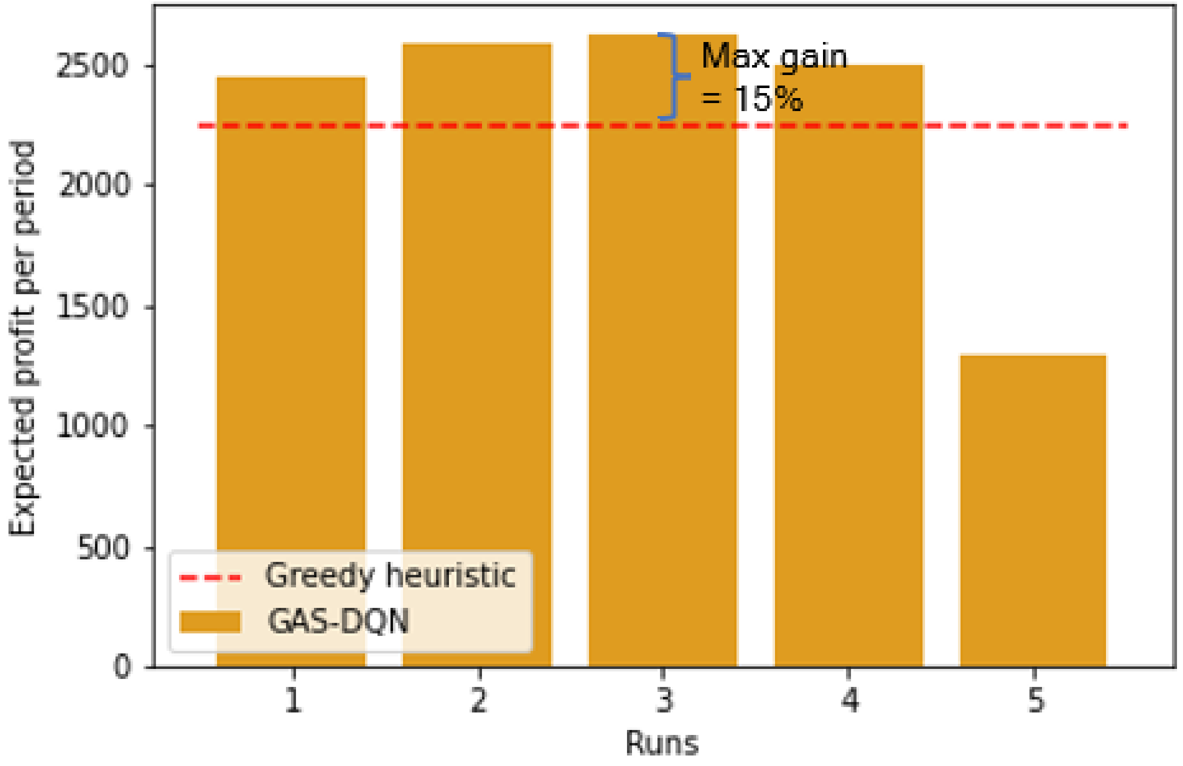
	Mapping function 1	Mapping function 2
Similarity	Take the outdating into account	
Difference	➢ Consider problem-specific factors (cost structure and risk distribution) ➢ Take more computing effort to find the best blend	➢ NOT consider problem-specific factors ➢ Easy to implement, less running time
Searching method	1. Maximum number of wine units from the last age class 2. Minimum sum of ages 3. Minimum total decay probability	1. Maximum number of wine units from the last age class 2. Take stock from the oldest age class to the youngest one





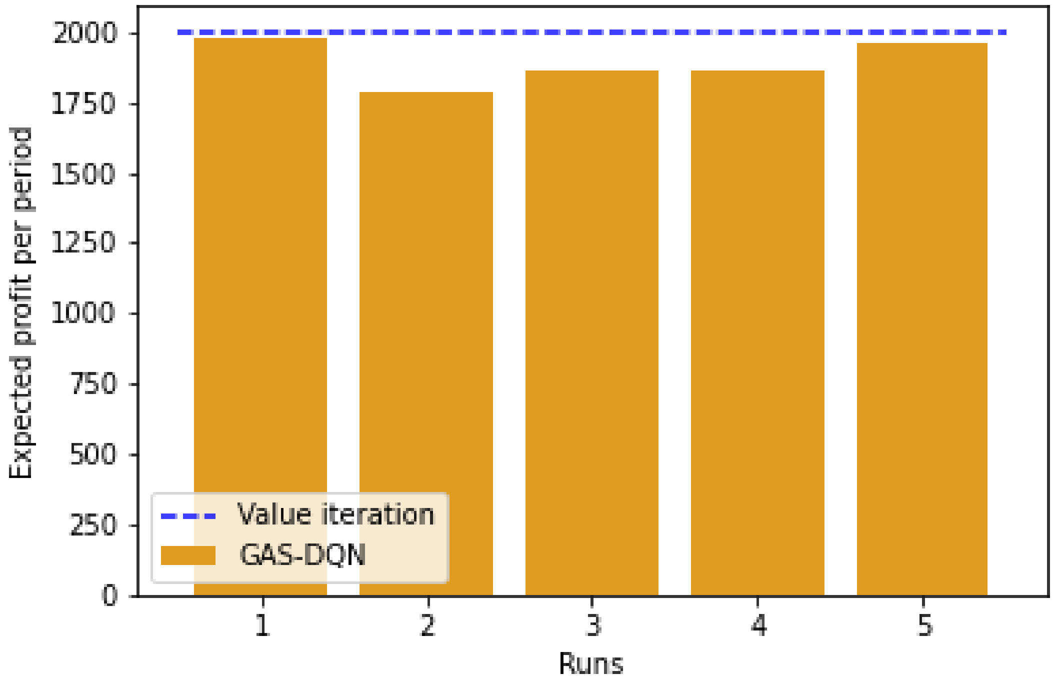
Experiment 2: Number of age classes (Scalability analysis)

Setting	$\lvert S \rvert$	$\lvert A \rvert$	Running time of GAS-DQN (s)
$\lvert I \rvert = 7$	823543	3456	8130
$\lvert I \rvert = 10$	10000000	17020	31033



Experiment 3: Difference between profit contributions of two branded products (parameter variation)

	$cb_1 = 333$ $cb_2 = 1000$	$cb_1 = 500$ $cb_2 = 600$
Avg. sales quantity of younger product	2.2	2.3
Underfulfillment (% of iterations)	54.0%	48.1%
Avg. sales quantity of older product	3.0	2.9
Underfulfillment (% of iterations)	0.09%	5.7%



Agenda

Limitations and further work

Limitation:

- DQN: number of nodes in output layer equal to number of actions
- GAS: more sub-spaces leads to higher computation time

Further work:

- Combine GAS with other value-based DRL frameworks
- Test on larger and more complex problems
- Benchmark with other methods

Thank you for listening!

References (1/2)

Boute RN, Gijsbrechts J, van Jaarsveld W, Vanvuchelen N (2021) Deep reinforcement learning for inventory control: A roadmap. European Journal of Operational Research. Forthcoming.

Chaharsooghi SK, Heydari J, Zegordi SH (2008) A reinforcement learning model for supply chain ordering management: An application to the beer game. Decision Support Systems. 45(4):949–959.

Civelek I, Karaesmen I, Scheller-Wolf A (2015) Blood platelet inventory management with protection levels. European Journal of Operational Research. 243(3):826–838.

Cohen MA (1976) Analysis of Single Critical Number Ordering Policies for Perishable Inventories. Operations Research. 24(4):726–741.

Duan Q, Liao TW (2013) A new age-based replenishment policy for supply chain inventory optimization of highly perishable products. International Journal of Production Economics. 145(2):658–671.

Farquhar G, Gustafson L, Lin Z, Whiteson S, Usunier N, Synnaeve G (2020) Growing Action Spaces. III HD, Singh A, eds. Proceedings of the 37th International Conference on Machine Learning, Proceedings of Machine Learning Research, 3040–3051.

Giannoccaro I, Pontrandolfo P (2002) Inventory management in supply chains: a reinforcement learning approach. International Journal of Production Economics. 78(2):153–161.

Gijsbrechts J, Boute RN, van Mieghem JA, Zhang D (2018) Can Deep Reinforcement Learning Improve Inventory Management? Performance and Implementation of Dual Sourcing-Mode Problems. SSRN Journal. Forthcoming.

Gutierrez-Alcoba A, Rossi R, Martin-Barragan B, Hendrix EM (2017) A simple heuristic for perishable item inventory control under non-stationary stochastic demand. International Journal of Production Research. 55(7):1885–1897.

Harsha P, Jagmohan A, Kalagnanam J, Quanz B, Singhvi D (2021) Math Programming based Reinforcement Learningfor Multi-Echelon Inventory Management. SSRN Journal. Forthcoming.

Kara A, Dogan I (2018) Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. Expert Systems with Applications. 91:150–158.

Kemmer L, Kleist H von, Rochebouët D de, Tziortziotis N, Read J (2018) Reinforcement learning for supply chain optimization. European Workshop on Reinforcement Learning.

References (2/2)

Meisheri H, Sultana NN, Baranwal M, Baniwal V, Nath S, Verma S, Ravindran B, Khadilkar H (2021) Scalable multi-product inventory control with lead time constraints using reinforcement learning. Neural Comput & Applic. Forthcoming.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, 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(7540):529–533.

Moor BJ de, Gijsbrechts J, Boute RN (2021) Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. European Journal of Operational Research. Forthcoming.

Nahmias S, Pierskalla WP (1976) A Two-Product Perishable/Nonperishable Inventory Problem. SIAM J. Appl. Math. 30(3):483–500.

Odoni AR (1969) On Finding the Maximal Gain for Markov Decision Processes. Operations Research. 17(5):857–860.

Oroojlooyjadid A, Nazari M, Snyder LV, Takáč M (2021) A Deep Q-Network for the Beer Game: Deep Reinforcement Learning for Inventory Optimization. M&SOM. Forthcoming.

Peng Sun, Kai Wang, Paul H. Zipkin (2014) Quadratic Approximation of Cost Functions in Lost Sales and Perishable Inventory Control Problems.

Seetharaman P, Wichern G, Pardo B, Le Roux J (2020) AutoClip: Adaptive Gradient Clipping for Source Separation Networks. 2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP) (IEEE).

Sultana N, Meisheri H, Baniwal V, Nath S, Ravindran B, Khadilkar H (2020) Reinforcement Learning for Multi-Product Multi-Node Inventory Management in Supply Chains.

Sutton RS, Barto AG (2018) Reinforcement learning. An introduction / Richard S. Sutton and Andrew G. Barto. Adaptive computation and machine learning (The MIT Press, Cambridge, Massachusetts).

Tom Schaul, John Quan, Ioannis Antonoglou, David Silver (2016) Prioritized Experience Replay.

van Hasselt H, Guez A, Silver D (2016) Deep Reinforcement Learning with Double Q-Learning. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, 2094–2100.

Vanvuchelen N, Gijsbrechts J, Boute R (2020) Use of Proximal Policy Optimization for the Joint Replenishment Problem. Computers in Industry. 119:103239.

Watkins CJCH (1989) Learning from delayed rewards.