# Policy-based DRL and Benchmarking

Willem van Jaarsveld - w.l.v.jaarsveld@tue.nl

Technische Universiteit
Eindhoven
University of Technology

26/8/2020

Where innovation starts

# Deep reinforcement learning



DRL is a set of numerical techniques with the goal of finding good **neural network policies** for sequential decision making problem.

Three ingredients:

- Markov Decision Processes
- Simulation
- Neural network policies

# MDP example: inventory control



Each period $t$:

1. order placed in period $t - \tau$ arrives and is added to on-hand inventory.
2. We place the order $a_t$ that will arrive in period $t + \tau$ (=action).
3. Random demand $D_t$ arrives
4. Demand is satisfied using on-hand inventory;
5. Excess demand is lost at **cost** $p$; remaining inventory **costs** $h$ per item.

# MDP example: inventory control



Each period $t$:

1. order placed in period $t - \tau$ arrives and is added to on-hand inventory.
2. We place the order $a_t$ that will arrive in period $t + \tau$ (=action).
3. Random demand $D_t$ arrives
4. Demand is satisfied using on-hand inventory;
5. Excess demand is lost at **cost** $p$; remaining inventory **costs** $h$ per item.

**state** of system in step 2: the onhand inventory $I$, and the orders in the pipeline. I.e. $s_t = (I; x_1, \ldots, x_{\tau-1})$.

# MDP example: inventory control



Each period $t$:

1. order placed in period $t - \tau$ arrives and is added to on-hand inventory.
2. We place the order $a_t$ that will arrive in period $t + \tau$ (=action).
3. Random demand $D_t$ arrives
4. Demand is satisfied using on-hand inventory;
5. Excess demand is lost at **cost** $p$; remaining inventory **costs** $h$ per item.

**state** of system in step 2: the onhand inventory $I$, and the orders in the pipeline. I.e. $s_t = (I; x_1, \ldots, x_{\tau-1})$.

Suppose we may only place orders $a_t \in \mathcal{A} = \{0, \ldots, \bar{a}\}$ (e.g. using bounds of Zipkin, 2008).

# Markov decision process - general case



Each period $t = 1, 2, \ldots$ the process/environment occupies a state, a decision is made, costs are incurred, and a transition is made to the next state.

Denote the state, action, and cost for period $t$ by $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ and $c_t$, respectively.

- $\mathcal{S}$ is the state space, and $\mathcal{A} = \{1, \ldots, m\}$ is the action space.
- $\mathcal{A}_s$ are the actions applicable in $s$; assume $\mathcal{A}_s \subseteq \mathcal{A}$.

# Markov decision process - general case



Each period $t = 1, 2, \ldots$ the process/environment occupies a state, a decision is made, costs are incurred, and a transition is made to the next state.

Denote the state, action, and cost for period $t$ by $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ and $c_t$, respectively.

- $\mathcal{S}$ is the state space, and $\mathcal{A} = \{1, \ldots, m\}$ is the action space.
- $\mathcal{A}_s$ are the actions applicable in $s$; assume $\mathcal{A}_s \subseteq \mathcal{A}$.

$\mathbf{P}(s_{t+1}|s_t, a_t)$ is the probability of ending up in $s_{t+1}$ when taking action $a_t$ in state $s_t$.

$C(s_t, a_t)$ denotes the cost of taking action $a_t$ in state $s_t$.

Following policy $\pi$ implies that $a_t = \pi(s_t)$ for each $t$. Optimal policy?!

# A simulation trajectory



Suppose $\tau = 4$, and we are in state $s_t = (I; x_1, x_2, x_3) = (2; 1, 0, 2)$.

- 2 items on-hand, and orders due to arrive in periods $t+1, t+2, t+3$ have size $1, 0, 2$, respectively.

# A simulation trajectory



Suppose $\tau = 4$, and we are in state $s_t = (I; x_1, x_2, x_3) = (2; 1, 0, 2)$.

- 2 items on-hand, and orders due to arrive in periods $t+1, t+2, t+3$ have size $1, 0, 2$, respectively.

We can simulate a given policy $\pi$:

Suppose $\pi(s_t) = a_t = 1$, and $D_t = 2$, then $s_{t+1} = (1; 0, 2, 1)$ and $c_t = h$

# A simulation trajectory



Suppose $\tau = 4$, and we are in state $s_t = (I; x_1, x_2, x_3) = (2; 1, 0, 2)$.

- 2 items on-hand, and orders due to arrive in periods $t+1, t+2, t+3$ have size $1, 0, 2$, respectively.

We can simulate a given policy $\pi$:

Suppose $\pi(s_t) = a_t = 1$, and $D_t = 2$, then $s_{t+1} = (1; 0, 2, 1)$ and $c_t = h$
Suppose $\pi(s_{t+1}) = a_{t+1} = 2$, and $D_{t+1} = 3$, then $s_{t+2} = (0; 2, 1, 2)$ and $c_{t+1} = 2p$.

# A simulation trajectory



Suppose $\tau = 4$, and we are in state $s_t = (I; x_1, x_2, x_3) = (2; 1, 0, 2)$.

- 2 items on-hand, and orders due to arrive in periods $t+1, t+2, t+3$ have size $1, 0, 2$, respectively.

We can simulate a given policy $\pi$:

Suppose $\pi(s_t) = a_t = 1$, and $D_t = 2$, then $s_{t+1} = (1; 0, 2, 1)$ and $c_t = h$
Suppose $\pi(s_{t+1}) = a_{t+1} = 2$, and $D_{t+1} = 3$, then $s_{t+2} = (0; 2, 1, 2)$ and $c_{t+1} = 2p$.
etc.

Goal: Determine a policy that can minimize long run expected costs.
Challenge: A policy $\pi$ sets an action for every state; there may be many states.

# Neural networks



A neural network is a parameterized function $\mathrm{N}_\theta$ from $\mathbb{R}^{\mathrm{N}}$ to $\mathbb{R}^{\mathrm{M}}$.

For any $\mathbf{x} \in \mathbb{R}^{\mathrm{N}}$, we have $\mathbf{y} = \mathrm{N}_\theta(\mathbf{x}) \in \mathbb{R}^{\mathrm{M}}$.

# Neural networks



A neural network is a parameterized function $N_\theta$ from $\mathbb{R}^N$ to $\mathbb{R}^M$.

For any $\mathbf{x} \in \mathbb{R}^N$, we have $\mathbf{y} = N_\theta(\mathbf{x}) \in \mathbb{R}^M$.

Example 1 (perceptron):

$$N_\theta(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$$

Perceptron parameters are $\theta = (A, \mathbf{b})$, with $A - M \times N$ matrix and $\mathbf{b} - M \times 1$ vector.

# The multilayer perceptron (MLP)



Composed of multiple operations or *layers*: denote the output of layer $l$ by $\psi^{[l]}$.

Layer/operation $l \in \{1, \ldots, L\}$ is parameterized by matrix $A^{[l]}$ and a vector $\mathbf{b}^{[l]}$

Thus $\theta = \{A^{[1]}, \mathbf{b}^{[1]}, \ldots, A^{[L]}, \mathbf{b}^{[L]}\}$.

# The multilayer perceptron (MLP)



Composed of multiple operations or *layers*: denote the output of layer $l$ by $\psi^{[l]}$.

Layer/operation $l \in \{1, \ldots, L\}$ is parameterized by matrix $A^{[l]}$ and a vector $\mathbf{b}^{[l]}$
Thus $\theta = \{A^{[1]}, \mathbf{b}^{[1]}, \ldots, A^{[L]}, \mathbf{b}^{[L]}\}$.

Output $\psi^{[l]}$ of layer $l$ is obtained as follows:

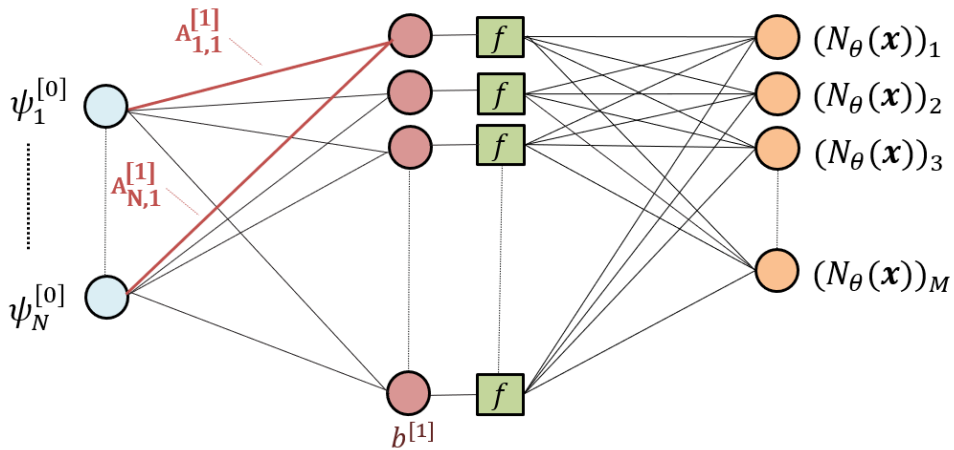
$$\psi^{[l]} = f^{[l]}(A^{[l]}\psi^{[l-1]} + \mathbf{b}^{[l]})$$

Here $f^{[l]}$ is the activation function of the layer, e.g. $f^{[l]}(\mathbf{x}) = \max(\mathbf{x}, 0)$.

The MLP is then defined by $N_\theta(\mathbf{x}) := \psi^{[L]}$, with $\psi^{[0]} := \mathbf{x}$.

**Input layer** **Hidden layer** **Output layer**

$$\psi_1^{[1]} = f^{[1]}\left(\sum_i A_{i,1}^{[1]}\psi_i^{[0]} + b_1^{[1]}\right)$$



# Neural network policies



$N_\theta : \mathbb{R}^N \to \mathbb{R}^M$. Suppose $\mathcal{S} \subseteq \mathbb{R}^N$, and $M = m = |\mathcal{A}|$. E.g. $s_t = (I; x_1, x_2, x_3) \in \mathbb{R}^4$

Then *for any* $\theta$, $N_\theta$ represents a policy:

$$\pi_\theta(s) = \arg\max_{a \in \mathcal{A}_s} [(N_\theta(s))_a]$$

# Neural network policies



$N_\theta : \mathbb{R}^N \to \mathbb{R}^M$. Suppose $\mathcal{S} \subseteq \mathbb{R}^N$, and $M = m = |\mathcal{A}|$. E.g. $s_t = (I; x_1, x_2, x_3) \in \mathbb{R}^4$

Then *for any* $\theta$, $N_\theta$ represents a policy:

$$\pi_\theta(s) = \arg\max_{a \in \mathcal{A}_s} [(N_\theta(s))_a]$$

**Alternatively,** use "soft arg-max" $\sigma(\cdot)$ to obtain stochastic policy:

$$\pi_\theta(a|s) = P(\pi_\theta(s) = a) = \sigma[N_\theta(s)]_a$$

**Alternatively,** in deep q− learning the neural networks approximate the optimal q values, i.e. we try to find $\theta$ such that $(N_\theta(s))_a \approx q_{\pi^*}(s, a)$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$. We need $N_\theta : \mathbb{R}^4 \to \mathbb{R}^3$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$. We need $N_\theta : \mathbb{R}^4 \to \mathbb{R}^3$.Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$;

Let $A = \begin{bmatrix} 1 & 2 & -2 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$, $A' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$, $\mathbf{b}' = \begin{bmatrix} 0 \\ 2 \\ 0.5 \end{bmatrix}$.

For any $s \in \mathbb{R}^4$, let $N_\theta(s) = A'\psi + \mathbf{b}'$ with $\psi = (As + \mathbf{b})^+$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$. We need $N_\theta : \mathbb{R}^4 \to \mathbb{R}^3$.Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$;

Let $A = \begin{bmatrix} 1 & 2 & -2 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$, $A' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$, $\mathbf{b}' = \begin{bmatrix} 0 \\ 2 \\ 0.5 \end{bmatrix}$.

For any $\mathbf{s} \in \mathbb{R}^4$, let $N_\theta(\mathbf{s}) = A'\psi + \mathbf{b}'$ with $\psi = (A\mathbf{s} + \mathbf{b})^+$.

E.g. $\mathbf{s} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$; then $\psi = \left( \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \right)^+ = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$ and $N_\theta(\mathbf{s}) = \begin{bmatrix} 2 \\ 3 \\ -0.5 \end{bmatrix}$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$. We need $N_\theta : \mathbb{R}^4 \to \mathbb{R}^3$.Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$;

Let $A = \begin{bmatrix} 1 & 2 & -2 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$, $A' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$, $\mathbf{b}' = \begin{bmatrix} 0 \\ 2 \\ 0.5 \end{bmatrix}$.

For any $s \in \mathbb{R}^4$, let $N_\theta(s) = A'\psi + \mathbf{b}'$ with $\psi = (As + \mathbf{b})^+$.

E.g. $s = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$; then $\psi = \left( \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \right)^+ = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$ and $N_\theta(s) = \begin{bmatrix} 2 \\ 3 \\ -0.5 \end{bmatrix}$.

Construct a policy as follows: $\pi_\theta(s) = \arg\max_{a \in \mathcal{A}_s} (N_\theta(s))_a$.

# Neural network policies: example



Example: suppose $\tau = 4$ and $\mathcal{A} = \{1, 2, 3\}$. We need $N_\theta : \mathbb{R}^4 \to \mathbb{R}^3$.Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$;

Let $A = \begin{bmatrix} 1 & 2 & -2 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & -1 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$, $A' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$, $\mathbf{b}' = \begin{bmatrix} 0 \\ 2 \\ 0.5 \end{bmatrix}$.

For any $s \in \mathbb{R}^4$, let $N_\theta(s) = A'\psi + \mathbf{b}'$ with $\psi = (As + \mathbf{b})^+$.

E.g. $s = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$; then $\psi = \left( \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \right)^+ = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$ and $N_\theta(s) = \begin{bmatrix} 2 \\ 3 \\ -0.5 \end{bmatrix}$.

Construct a policy as follows: $\pi_\theta(s) = \arg\max_{a \in \mathcal{A}_s} (N_\theta(s))_a$.Thus $\pi_\theta(s) = 2$.

# DRL: Optimizing $\theta$



Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$, with

$$A = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, A' = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}, \mathbf{b}' = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}.$$

For any $\mathbf{s} \in \mathbb{R}^4$, let $N_\theta(\mathbf{s}) = A'\psi + \mathbf{b}'$ with $\psi = (A\mathbf{s} + \mathbf{b})^+$.

$\pi_\theta(\mathbf{s}) = \arg\max_{a \in \mathcal{A}_\mathbf{s}} (N_\theta(\mathbf{s}))_a$

For **any** $\theta$, i.e. any $A, \mathbf{b}, A', \mathbf{b}'$, $\pi_\theta(\cdot)$ is an ordering policy for our lost sales inventory problem with leadtime $\tau = 4$.

# DRL: Optimizing $\theta$



Let $\theta = (A, \mathbf{b}, A', \mathbf{b}')$, with

$$A = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}, A' = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}, \mathbf{b}' = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}.$$

For any $\mathbf{s} \in \mathbb{R}^4$, let $N_\theta(\mathbf{s}) = A'\psi + \mathbf{b}'$ with $\psi = (A\mathbf{s} + \mathbf{b})^+$.

$\pi_\theta(\mathbf{s}) = \arg\max_{a \in \mathcal{A}_\mathbf{s}} (N_\theta(\mathbf{s}))_a$

For **any** $\theta$, i.e. any $A, \mathbf{b}, A', \mathbf{b}'$, $\pi_\theta(\cdot)$ is an ordering policy for our lost sales inventory problem with leadtime $\tau = 4$.

Deep reinforcement learning is finding $\theta$ (i.e. $A, \mathbf{b}, A', \mathbf{b}'$) such that $\pi_\theta$ is a good policy.

# Positioning DRL in literature



Equivalent nomenclature: Approximate dynamic programming, Neuro-dynamic programming, reinforcement learning.

Parametric policy function approximations have a long history (see e.g. Powell 2019).

Adopting deep neural networks as policy function approximations resulted in breakthroughs in playing Atari games (A3C) and go, chess and shogi (AlphaZero, Silver et al., 2018).

# Positioning DRL in literature



Equivalent nomenclature: Approximate dynamic programming, Neuro-dynamic programming, reinforcement learning.

Parametric policy function approximations have a long history (see e.g. Powell 2019).

Adopting deep neural networks as policy function approximations resulted in breakthroughs in playing Atari games (A3C) and go, chess and shogi (AlphaZero, Silver et al., 2018).

Since then, everybody talks about *Deep Reinforcement Learning*.

# DRL algorithms



| Algorithm | Role(s) of NN | "Model-based" | "Model-free" |
|---|---|---|---|
| DQN | q-value estimation | | ✓ |
| REINFORCE | policy repr. | | ✓ |
| A3C | policy repr. (w. critic) | | ✓ |
| TRPO | policy repr. (w. critic) | | ✓ |
| PPO | policy repr. (w. critic) | | ✓ |
| AlphaZero | hybrid | ✓ | |
| RL as classification[1] | policy representation | ✓ | |

[1] Lagoudakis & Parr (2003)

# DRL algorithms



| Algorithm | Role(s) of NN | "Model-based" | "Model-free" |
|---|---|---|---|
| DQN | q-value estimation | | ✓ |
| REINFORCE | policy repr. | | ✓ |
| A3C | policy repr. (w. critic) | | ✓ |
| TRPO | policy repr. (w. critic) | | ✓ |
| PPO | policy repr. (w. critic) | | ✓ |
| AlphaZero | hybrid | ✓ | |
| RL as classification[1] | policy representation | ✓ | |

[1] Lagoudakis & Parr (2003)

# DRL algorithms



| Algorithm | Role(s) of NN | "Model-based" | "Model-free" |
|---|---|---|---|
| DQN | q-value estimation | | ✓ |
| REINFORCE | policy repr. | | ✓ |
| A3C | policy repr. (w. critic) | | ✓ |
| TRPO | policy repr. (w. critic) | | ✓ |
| PPO | policy repr. (w. critic) | | ✓ |
| AlphaZero | hybrid | ✓ | |
| RL as classification[1] | policy representation | ✓ | |

[1] Lagoudakis & Parr (2003)

# DRL algorithms



| Algorithm | Role(s) of NN | "Model-based" | "Model-free" |
|---|---|---|---|
| DQN | q-value estimation | | ✓ |
| REINFORCE | policy repr. | | ✓ |
| A3C | policy repr. (w. critic) | | ✓ |
| TRPO | policy repr. (w. critic) | | ✓ |
| PPO | policy repr. (w. critic) | | ✓ |
| AlphaZero | hybrid | ✓ | |
| RL as classification[1] | policy representation | ✓ | |

[1] Lagoudakis & Parr (2003)

# DRL algorithms



| Algorithm | Role(s) of NN | "Model-based" | "Model-free" |
|---|---|---|---|
| DQN | q-value estimation | | ✓ |
| REINFORCE | policy repr. | | ✓ |
| A3C | policy repr. (w. critic) | | ✓ |
| TRPO | policy repr. (w. critic) | | ✓ |
| PPO | policy repr. (w. critic) | | ✓ |
| AlphaZero | hybrid | ✓ | |
| RL as classification[1] | policy representation | ✓ | |

All these algorithms:

1. Aim to find a policy $\pi_\theta$ that performs well on the MDP environment.
2. Update $\theta$ through lot's of simulation/interaction with the system.

[1] Lagoudakis & Parr (2003)

# Model-free algorithms



DQN, A3C, PPO:

- Initial successes: ATARI, MuJoCO.
- Also employed by DRL researchers in inventory control

# Model-free algorithms



DQN, A3C, PPO:

- Initial successes: ATARI, MuJoCO.
- Also employed by DRL researchers in inventory control

Rough idea:

- Simulate underlying MDP while using some initial neural network policy, yielding trajectories $s_t, a_t, c_t, s_{t+1}, a_{t+1}, c_t, \ldots$.
- Improve neural network parameters gradually using info gathered from trajectories.
- In particular, merit of taking action $a_t$ in state $s_t$ is gauged based on the trajectory costs incurred in periods $t, t+1, t+2, \ldots$.
- Update $\theta$ to increase/decrease $(N_\theta(s_t))_{a_t}$, depending on whether costs are low/high relative to some benchmark.

# Results for lost sales

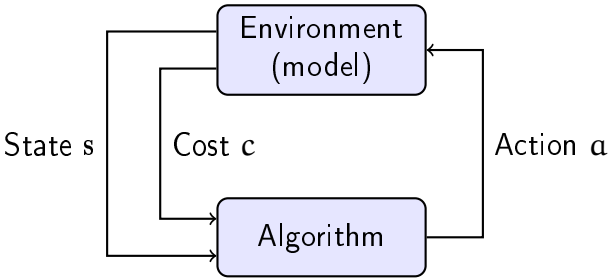


Seminal work of Gijsbrechts, Boute, Van Mieghem, Zhang (2018, SSRN; 2022, MSOM):

| | | Poisson | | |
|---|---|---|---|---|
| | | Lead-time $\tau$ | | |
| Penalty | Policy | 2 | 3 | 4 |
| $p = 4$ | Base-stock | 5.5% | 8.2% | 9.9% |
| | A3C | 3.2% | 3.0% | 6.7% |
| $p = 9$ | Base-stock | 3.7% | 5.1% | 6.4% |
| | A3C | 4.8% | 3.1% | 3.4% |

# “Model-free” DRL (A3C, PPO)





# Observations



If a state $s$ and optimal action $a$ is followed by unfavorable random events, then the algorithm may wrongly deem the action as *unattractive*.

Conversely, if state $s$ and mediocre/bad action $a$ is followed by favorable events, then the algorithm may wrongly deem the action as *attractive*.

# Observations



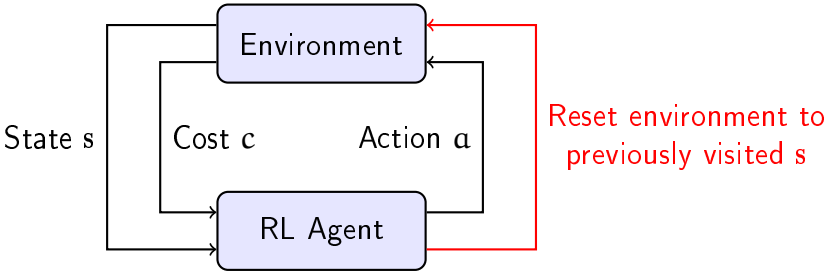
If a state $s$ and optimal action $a$ is followed by unfavorable random events, then the algorithm may wrongly deem the action as *unattractive*.

Conversely, if state $s$ and mediocre/bad action $a$ is followed by favorable events, then the algorithm may wrongly deem the action as *attractive*.

This is especially problematic in highly stochastic environments that arise in online decision making for OM, and for such environments other learning approaches may be advisable.

# Model-based algorithms

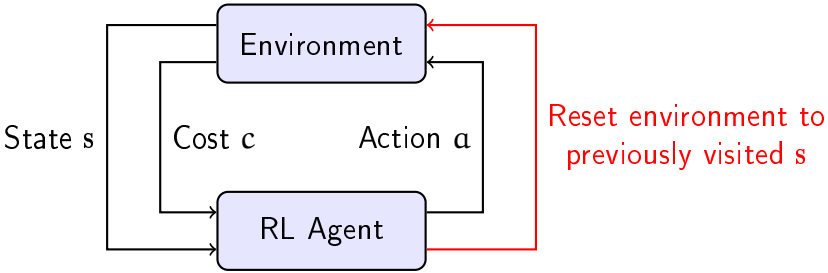





Model-based DRL approaches date back to Lagoudakis and Parr (2003). AlphaZero (2017) was big breakthrough.

# Model-based algorithms






Model-based DRL approaches date back to Lagoudakis and Parr (2003). AlphaZero (2017) was big breakthrough.

*Deep controlled learning* is a variant specifically designed for online decision making in operations management (Temizöz et al., 2023): https://arxiv.org/abs/2011.15122.



Stochastic single-player games

Deterministic 2-player games

# Value function of heuristic $\pi_0$



- Consider an MDP that eventually always reaches some terminal state. (Like the airline seat example you solved as preparation.)
- Let $V_{\pi_0}(s)$ denote the expected total return of being in state $s$, and following policy $\pi_0$ until reaching a terminal state.
- This *value function* $V_{\pi_0}(s), s \in \mathcal{S}$ for policy $\pi_0$ solves:

$$V_{\pi_0}(s) = C(s, \pi_0(s)) + \sum_{s' \in \mathcal{S}} P(s'|s, \pi_0(s)) V_{\pi_0}(s')$$

- When $|\mathcal{S}|$ is $\leqslant 10^6 - 10^7$, numerical techniques can typically solve this.

# Q– values (after action values)



- Now suppose we are in state $s$, and we take action $a \in \mathcal{A}_s$ (with $a \neq \pi_0(s)$) *once*, and revert to $\pi_0$ afterwards.
- The associated value function is

$$q_{\pi_0}(s, a) = C(s, a) + \sum_{s' \in \mathcal{S}} P(s'|s, a) V_{\pi_0}(s')$$

# Exact policy iteration



- Let:

$$q_{\pi_0}(s, a) = C(s, a) + \alpha \sum_{s' \in \mathcal{S}} P(s'|s, a) V_{\pi_0}(s')$$

  with $V_{\pi_0}(s')$ the values associated with $\pi_0$.
- Define $\pi_1$ such that $\pi_1(s) \in \arg\min_{a \in \mathcal{A}(s)} q_{\pi_0}(s, a)$ for all $s$.
- Define $\pi_2$ such that $\pi_2(s) \in \arg\min_{a \in \mathcal{A}(s)} q_{\pi_1}(s, a)$ for all $s$.
- These are two exact policy iteration steps.
- An appropriate continuation of such steps converges to the optimal policy.

# Deep controlled learning



Starting from some policy $\pi_0$, we make a few *approximate policy iteration steps*.

Approximate policy improvement step from $\pi_i$:

1. Collect data points $\mathcal{K}_i = \{(s_k, a_k) | k \in \{1, \ldots, K\}\}$.
   - Somehow select some subset of states $s_k$ for $k \in \{1, \ldots, K\}$, e.g. by simulating $\pi_i$.
   - Use simulation-based one-step policy improvement over $\pi_i$ to find $a_k \approx \arg\min_{a \in \mathcal{A}(s)} q_{\pi_i}(s, a)$.
2. Learn neural network parameters $\theta(\mathcal{K}_i)$ from $\mathcal{K}_i$ using supervised learning.
3. Set $\pi_{i+1} = \pi_{\theta(\mathcal{K}_i)}$

# Deep controlled learning



Starting from some policy $\pi_0$, we make a few *approximate policy iteration steps*.

Approximate policy improvement step from $\pi_i$:

1. Collect data points $\mathcal{K}_i = \{(s_k, a_k) | k \in \{1, \ldots, K\}\}$.
   - Somehow select some subset of states $s_k$ for $k \in \{1, \ldots, K\}$, e.g. by simulating $\pi_i$.
   - Use simulation-based one-step policy improvement over $\pi_i$ to find $a_k \approx \arg\min_{a \in \mathcal{A}(s)} q_{\pi_i}(s, a)$.
2. Learn neural network parameters $\theta(\mathcal{K}_i)$ from $\mathcal{K}_i$ using supervised learning.
3. Set $\pi_{i+1} = \pi_{\theta(\mathcal{K}_i)}$

# Deep controlled learning



Starting from some policy $\pi_0$, we make a few *approximate policy iteration steps*.

Approximate policy improvement step from $\pi_i$:

1. Collect data points $\mathcal{K}_i = \{(s_k, a_k) | k \in \{1, \ldots, K\}\}$.
   - Somehow select some subset of states $s_k$ for $k \in \{1, \ldots, K\}$, e.g. by simulating $\pi_i$.
   - Use simulation-based one-step policy improvement over $\pi_i$ to find $a_k \approx \arg\min_{a \in \mathcal{A}(s)} q_{\pi_i}(s, a)$.
2. Learn neural network parameters $\theta(\mathcal{K}_i)$ from $\mathcal{K}_i$ using supervised learning.
3. Set $\pi_{i+1} = \pi_{\theta(\mathcal{K}_i)}$

# Collecting a data point



Let $\xi$ denote a composite random variable comprising all randomness that may influence a trajectory, i.e. $\xi = (W_1, W_2, \ldots, W_T)$ in Martijn's notation.

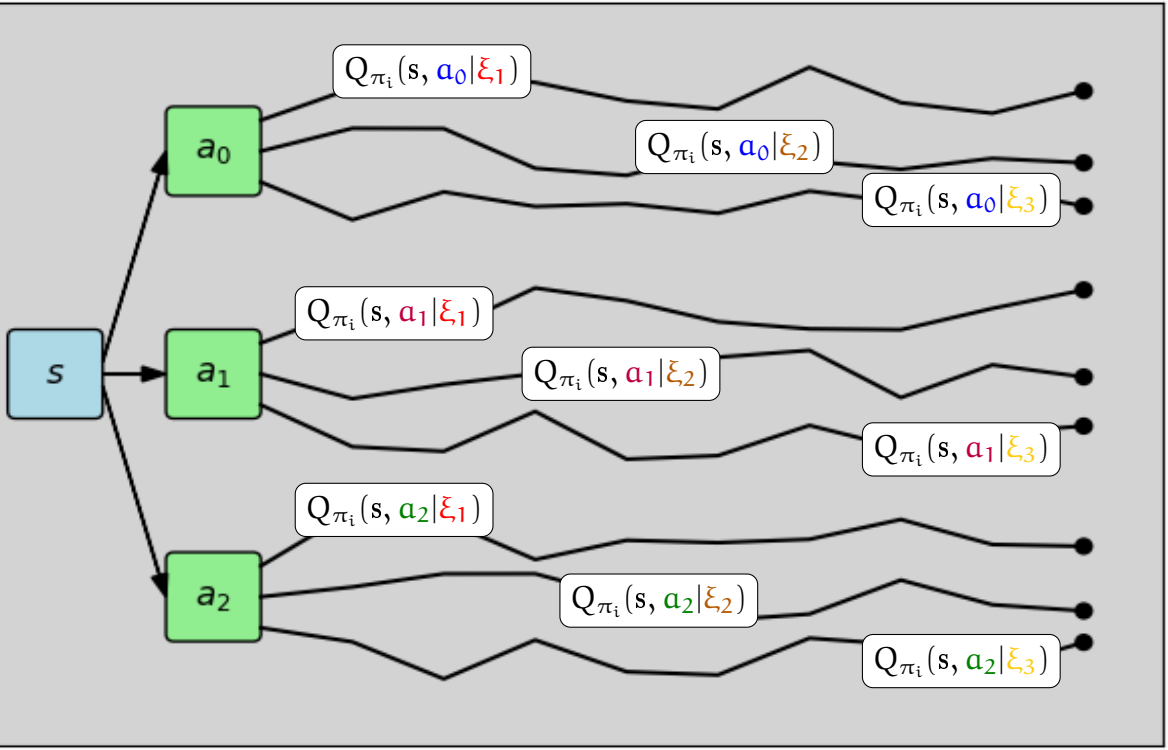
For the lost sales inventory model we would have $\xi = (D_t, D_{t+1}, \ldots, D_T)$.

Denote (random) trajectory cost for a random trajectory starting in $s$ with action $a$, and following $\pi$ afterwards by $Q_\pi(s, a|\xi)$.

When we always reach a terminal state, $\mathbf{E}_\xi(Q_{\pi_i}(s, a|\xi)) = q_{\pi_i}(s, a)$. (Otherwise, we run for $m$ steps to approximate.)
We draw several replications $\xi_1, \xi_2, \ldots, \xi_n$, and set:

$$a_k = \arg\min_{a \in A(s)} \frac{1}{n} \sum_{j=1}^{n} Q_{\pi_i}(s_k, a|\xi_n)$$

$a_k$ improves upon $\pi_i(s_k)$ and $(s_k, a_k)$ is used to train the next generation.


s
$a_0$
$a_1$
$a_2$
$Q_{\pi_i}(s, a_0|\xi_1)$
$Q_{\pi_i}(s, a_0|\xi_2)$
$Q_{\pi_i}(s, a_0|\xi_3)$
$Q_{\pi_i}(s, a_1|\xi_1)$
$Q_{\pi_i}(s, a_1|\xi_2)$
$Q_{\pi_i}(s, a_1|\xi_3)$
$Q_{\pi_i}(s, a_2|\xi_1)$
$Q_{\pi_i}(s, a_2|\xi_2)$
$Q_{\pi_i}(s, a_2|\xi_3)$

# Why control variance?



Is $q_\pi(s,a) - q_\pi(s,a')$ bigger or smaller than zero?

Let $\xi_i$ for $i = 1,\ldots,r$ denote independent replications of $\xi$, and let

$$X = \frac{1}{r}\sum_{i=1}^{r} Q_\pi(s,a|\xi_i) - Q_\pi(s,a'|\xi_i)$$

We have $\mathbf{E}[X] = q_\pi(s,a) - q_\pi(s,a')$ and

$$\text{var}[X] = \frac{1}{r}\big[\,\text{var}[Q_\pi(s,a|\xi)] + \text{var}[Q_\pi(s,a'|\xi)] - 2\,\text{cov}[Q_\pi(s,a|\xi), Q_\pi(s,a'|\xi)]\big].$$

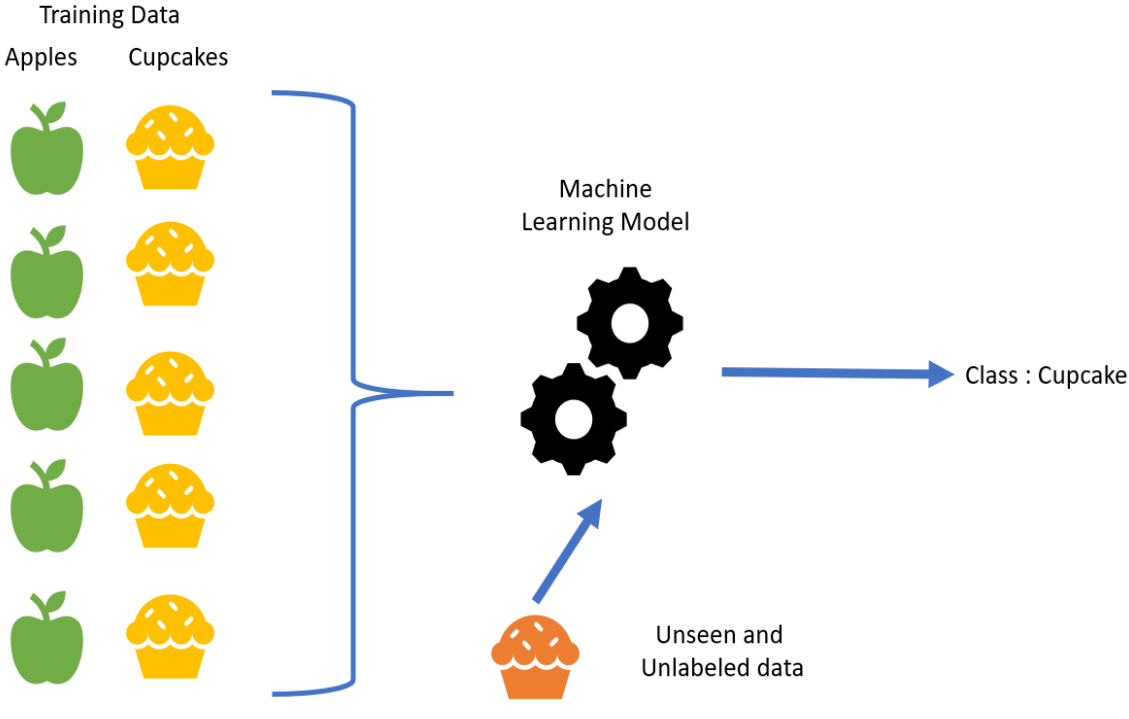
# Deep controlled learning



Starting from some policy $\pi_0$, we make a few *approximate policy iteration steps*.

Approximate policy improvement step from $\pi_i$:

1. Collect data points $\mathcal{K}_i = \{(s_k, a_k) | k \in \{1, \ldots, K\}\}$.
   - Somehow select some subset of states $s_k$ for $k \in \{1, \ldots, K\}$, e.g. by simulating $\pi_i$.
   - Use simulation-based one-step policy improvement over $\pi_i$ to find $a_k \approx \arg\min_{a \in \mathcal{A}(s)} q_{\pi_i}(s, a)$.
2. Learn neural network parameters $\theta(\mathcal{K}_i)$ from $\mathcal{K}_i$ using supervised learning.
3. Set $\pi_{i+1} = \pi_{\theta(\mathcal{K}_i)}$


Training Data
Apples
Cupcakes
Machine
Learning Model
Class : Cupcake
Unseen and
Unlabeled data

# Training the neural network



Minimize the average loss over samples $\mathcal{K}_i$:

$$\mathcal{L}_\theta(\mathcal{K}) := \frac{1}{|\mathcal{K}_i|} \sum_{(s,a') \in \mathcal{K}} l(s, a'|\theta). \tag{1}$$

Per sample, we use cross-entropy loss:

$$l(s, a'|\theta) := -\log \frac{\exp\left(N_\theta(s)[a']\right)}{\sum_{a' \in \mathcal{A}_s} \exp\left(N_\theta(s)[a']\right)}, \tag{2}$$

($N_\theta(s)[a]$ represents the neural network's predicted probability for action $a$. )

# DCL results



# DCL results



| Study | Type | Problem | #SKUs | #locations |
|---|---|---|---|---|
| Dmitrochenko (2020) | Master thesis | Fully connected network | 1 | 5 |
| Gijsbrechts et al. (2022) | Journal article | Distribution system | 1 | 11 |
| Van Hezewijk et al. (2022) | Journal article | Capacitated lot sizing | 10 | - |
| Vanvuchelen et al. (2023a) | Journal article | Fully connected network | 1 | 10 |
| Vanvuchelen et al. (2023b) | Preprint | Joint replenishment | 40 | 1 |
| This study | Master thesis | Fully connected network | 10,000 | 60 |

# Proper benchmarking



- Lost sales
- Perishable inventory control
- Stochastic leadtimes with order-crossing

| | | Poisson | | | | | | Geometric | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Lead time $\tau$ | | | | | | Lead time $\tau$ | | | | | |
| p | Policy | 2 | 3 | 4 | 6 | 8 | 10 | 2 | 3 | 4 | 6 | 8 | 10 |
| 4 | BSP | 5.5% | 8.2% | 9.9% | 5.51 | 5.72 | 5.86 | 4.5% | 6.4% | 7.8% | 11.86 | 12.12 | 12.31 |
| | CBS | 0.2% | 0.7% | 1.5% | 5.03 | 5.19 | 5.27 | 0.8% | 0.4% | 0.8% | 10.91 | 10.96 | 10.98 |
| | M-2 | 0.2% | 0.8% | 1.9% | 5.05 | 5.20 | 5.31 | 0.5% | 1.2% | 1.7% | 11.08 | 11.27 | 11.40 |
| | A3C | 3.2% | 3.0% | 6.7% | – | – | – | – | – | – | – | – | – |
| | DCL | 0.00% | 0.03% | 0.04% | *4.87* | *4.95* | 5.04 | 0.02% | 0.02% | *0.02%* | *10.75* | *10.82* | 10.91 |
| | DCL′ | *0.00%* | *0.01%* | *0.03%* | *4.87* | *4.95* | *5.00* | *0.01%* | *0.01%* | *0.02%* | *10.75* | *10.82* | *10.87* |
| 9 | BSP | 3.7% | 5.1% | 6.4% | 7.90 | 8.32 | 8.63 | 3.1% | 4.6% | 5.8% | 18.53 | 19.18 | 19.68 |
| | CBS | 0.5% | 1.4% | 1.0% | 7.26 | 7.55 | 7.77 | 0.8% | 0.8% | 0.9% | 17.35 | 17.68 | 17.88 |
| | M-2 | 0.2% | 0.6% | 1.2% | 7.43 | 7.77 | 8.08 | 0.5% | 1.1% | 1.9% | 17.75 | 18.39 | 18.89 |
| | A3C | 4.8% | 3.1% | 3.4% | – | – | – | – | – | – | – | – | – |
| | DCL | *0.00%* | 0.01% | 0.04% | 7.23 | 7.52 | 7.85 | 0.02% | *0.01%* | 0.01% | 17.16 | 17.52 | 17.75 |
| | DCL′ | *0.00%* | *0.00%* | *0.03%* | *7.22* | *7.49* | *7.68* | *0.01%* | *0.01%* | *0.01%* | *17.13* | *17.48* | *17.71* |
| 19 | BSP | 2.3% | 2.9% | 3.9% | 10.20 | 10.90 | 11.48 | 2.0% | 3.0% | 3.9% | 25.54 | 26.81 | 27.82 |
| | CBS | 0.8% | 0.5% | 0.7% | 9.80 | 10.35 | 10.66 | 0.8% | 1.0% | 1.4% | 24.49 | 25.38 | 25.98 |
| | M-2 | 0.1% | 0.4% | 0.8% | 9.85 | 10.53 | 11.09 | 0.3% | 0.8% | 1.4% | 24.93 | 26.21 | 27.27 |
| | DCL | 0.01% | 0.02% | 0.06% | 9.66 | 10.20 | 10.82 | *0.01%* | *0.01%* | 0.03% | 24.20 | 25.10 | 25.81 |
| | DCL′ | *0.00%* | *0.01%* | *0.02%* | *9.64* | *10.17* | *10.57* | *0.01%* | *0.01%* | *0.01%* | *24.19* | *25.09* | *25.73* |
| 39 | BSP | 0.9% | 1.8% | 2.5% | 12.38 | 13.39 | 14.24 | 1.3% | 2.0% | 2.6% | 32.69 | 34.47 | 36.25 |
| | CBS | 0.3% | 0.4% | 0.8% | 12.08 | 12.94 | 13.71 | 0.3% | 1.1% | 1.4% | 31.86 | 33.97 | 35.64 |
| | M-2 | 0.1% | 0.3% | 0.4% | 12.11 | 13.09 | 13.93 | 0.2% | 0.5% | 0.9% | 32.12 | 34.12 | 35.82 |
| | DCL | *0.00%* | 0.04% | 0.10% | 11.97 | 12.82 | 14.22 | *0.01%* | 0.03% | 0.03% | 31.49 | 33.11 | 34.40 |
| | DCL′ | *0.00%* | *0.01%* | *0.02%* | *11.94* | *12.81* | *13.48* | *0.01%* | *0.02%* | *0.02%* | *31.45* | *33.07* | *34.33* |

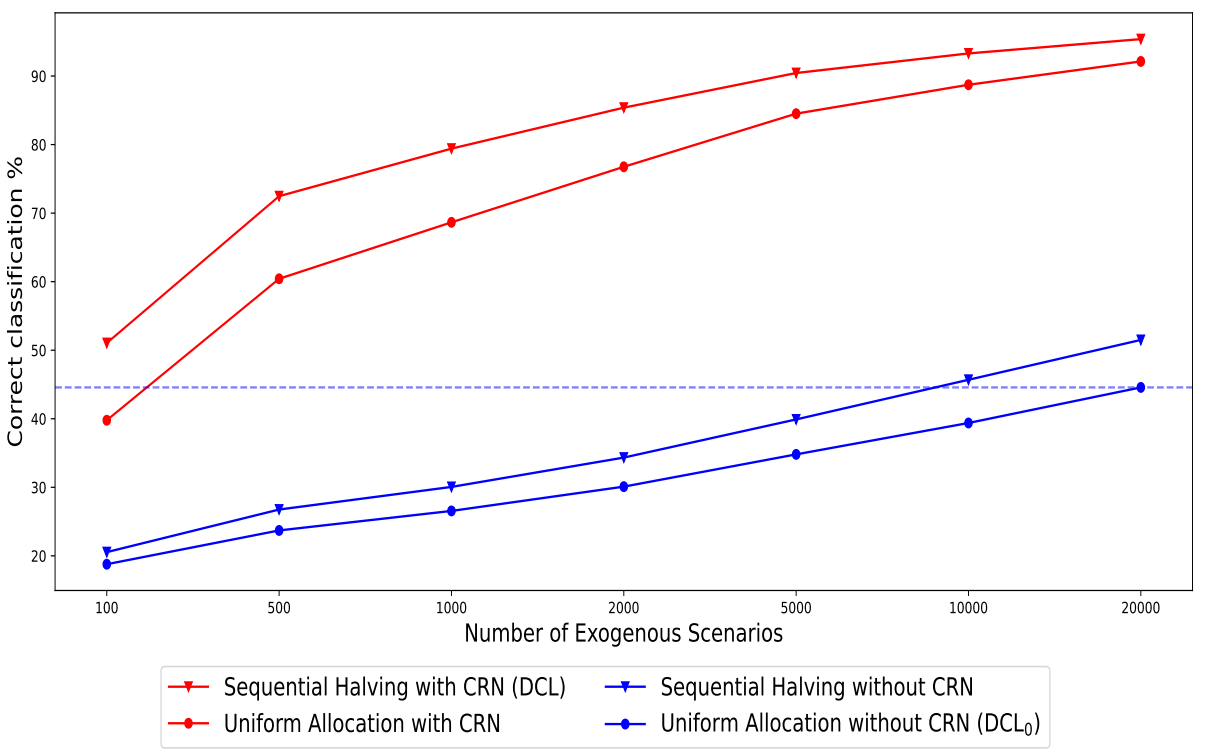
# DQN and ProBSP combination

- Check the impact of the penalty on the learned policy

| M | Lead Time | MTTF | a | ζ | ProBSP | DQN | DQN + RS (Training 1) | DQN + RS (Training 2) |
|---|---|---|---|---|---|---|---|---|
| 30 | 1 | 20 | 1.0 | 0.01 | 1.79 | 2.04 | 2.44 | 1.75 |
| 30 | 1 | 20 | 1.0 | 0.005 | 1.79 | 2.04 | 1.75 | 1.717 |
| 30 | 1 | 20 | 1.0 | 0.001 | 1.79 | 2.04 | 1.84 | 1.83 |
| 30 | 1 | 20 | 1.0 | 0.0005 | 1.79 | 2.04 | 1.99 | 1.96 |
| 30 | 1 | 20 | 1.0 | 0.0001 | 1.79 | 2.04 | 1.9 | 1.88 |
| 30 | 1 | 20 | 1.0 | 0.00005 | 1.79 | 2.04 | 1.98 | - |

- DQN is improved but still far from the DCL's improvement

| M | Lead Time | MTTF | a | ProBSP | DQN | DQN + RS | DCL | DCLPro |
|---|---|---|---|---|---|---|---|---|
| 30 | 1 | 20 | 1.0 | 1.79 | 2.04 | 1.85 | 1.92 | 1.56 |


Correct classification %
90
80
70
60
50
40
30
20
100
500
1000
2000
5000
10000
20000
Number of Exogenous Scenarios
Sequential Halving with CRN (DCL)
Uniform Allocation with CRN
Sequential Halving without CRN
Uniform Allocation without CRN (DCL0)

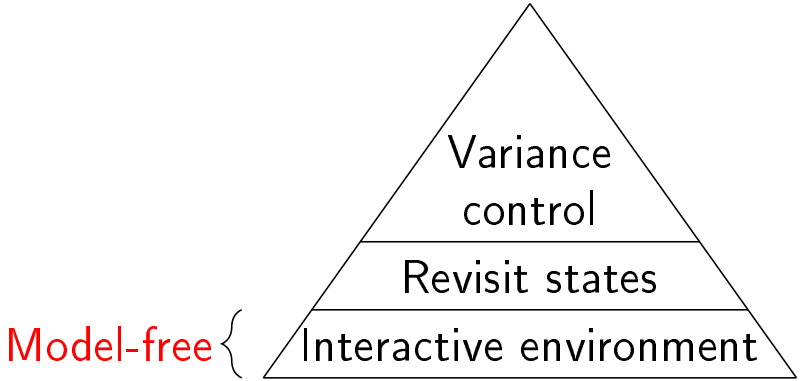
# Summary results



| | Lost Sales | | | Perishable | | | Random Lead Times | | |
|---|---|---|---|---|---|---|---|---|---|
| | DCL | $DCL_0$ | $\pi_{heur}$ | DCL | $DCL_0$ | $\pi_{heur}$ | DCL | $DCL_0$ | $\pi_{heur}$ |
| Optimality gap | 0.02% | 1.2% | 0.8% | 0.03% | 0.8% | 4.1% | 0.1% | 0.8% | 7.1% |
| BSP gap (large instances) | -6.2% | 2.8% | -5.4% | -9.0% | -7.8% | -4.3% | -12.7% | -12.0% | -7.3% |
| BSP gap (all instances) | -5.6% | -1.3% | -4.8% | -8.7% | -7.8% | -4.6% | -23.9% | -23.1% | -19.5% |

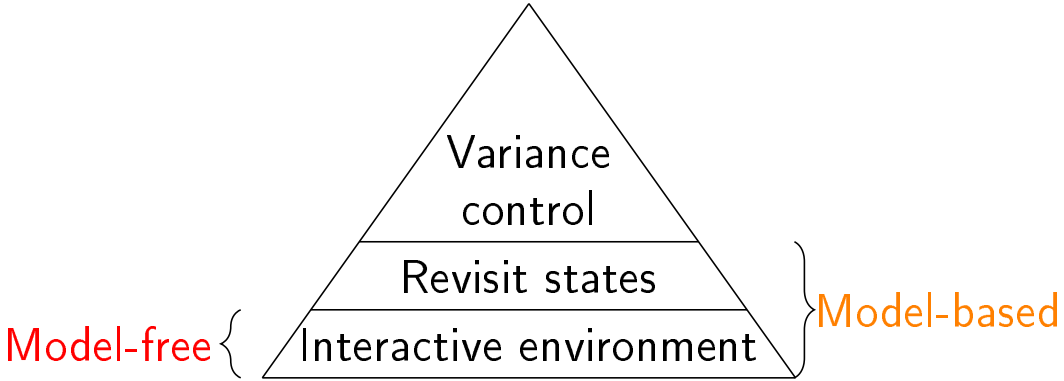
# Conclusions





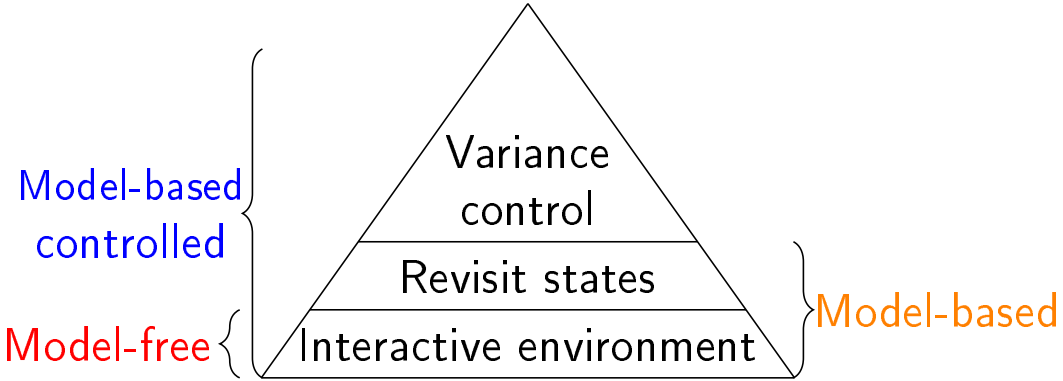
# Conclusions





# Conclusions





# DynaPlex



DynaPlex is a software project for DRL/ADP, focused on problems arising in logistics, operations management, and other settings with highly stochastic sequential decision problems.

# DynaPlex



DynaPlex is a software project for DRL/ADP, focused on problems arising in logistics, operations management, and other settings with highly stochastic sequential decision problems.

DynaPlex is also a DinaLog project by Martijn Mes, Willem van Jaarsveld, Remco Dijkman, Yingqian Zhang, Maria Iacob, Fabian Akkerman, Tarkan Temizoz, Luca Begnardi, Riccardo LoBianco.

# Why DynaPlex?



DRL for bigger problems requires major computational resources (cf. AlphaZero), and DynaPlex supports and simplifies the use of such resources, especially the Snellius supercomputer.

Publishing in OM requires

- benchmarking vs optimal solution (even if only for toy variant of problem).
- benchmarking vs heuristic policies.
- beating well thought-out benchmarks, and thus very capable DRL algorithms.

DynaPlex supports all this via a single API that allows you to plug in your MDP definition.

# MDP variable definition



```
namespace lost_sales /*must be consistent everywh
{
    class MDP
    {
    public:
        //MDP variables:
        double p, h;
        int64_t leadtime;
        int64_t MaxOrderSize;
        int64_t MaxSystemInv;
        DynaPlex::DiscreteDist demand_dist;

        //This is optional, if you want the disco
        //The static MDP information returned by
        //if not provided, defaults to 1.0.
        double  discount_factor;
```

# State definition



```
//A state is a struct (or class) that represents state infor
struct State {
    DynaPlex::StateCategory cat;
    Queue<int64_t> state_vector;
    int64_t total_inv;

    //declaration; for definition see mdp.cpp:
    DynaPlex::VarGroup ToVarGroup() const;
    //Defaulting this does not always work. It can be remove
    bool operator==(const State& other) const = default;
};
```

# Action transitions



```
double MDP::ModifyStateWithAction(State& state, int64_t action) const
{
    if (!IsAllowedAction(state, action))
        throw DynaPlex::Error("Lost Sales: action not allowed: state.total_inv: " +

    state.state_vector.push_back(action);
    state.total_inv += action;
    state.cat = StateCategory::AwaitEvent();
    return 0.0;
}
```

# Event transitions



```
double MDP::ModifyStateWithEvent(State& state,const MDP::Event& event) const
{
    state.cat= StateCategory::AwaitAction();
    auto onHand = state.state_vector.pop_front();//Length is leadtime again.
    if (onHand > event)
    {//There is sufficient inventory. Satisfy order and incur holding costs
        onHand -= event;
        state.total_inv -= event;
        state.state_vector.front() += onHand;
        return onHand * h;
    }
    else
    {
        state.total_inv -= onHand;
        return (event - onHand) * p;
    }
}
```

# Training loop (python)



```python
from dp.loader import DynaPlex as dp
sys.path.remove(parent_directory)

mdp = dp.get_mdp(id="lost_sales",
                 p=9.0, h=1.0,
                 leadtime=3,
                 demand_dist={"type": "poisson", "mean": 3.0})


base_policy = mdp.get_policy("base_stock")
dcl = dp.get_dcl(mdp, None, M=500, num_gens=1)
dcl.train_policy()
trained_policies = dcl.get_policies()
comparer = dp.get_comparer(mdp, number_of_trajectories=100, rng_seed=12)

comparison = comparer.compare(trained_policies)
result = [( item['mean']) for item in comparison]
print(result)
```

# DynaPlex in this course



We recommend the use of DynaPlex in the course, especially if your problem is in the inventory or maintenance optimization domain:

- Examples in our field
- Performant algorithms that beat benchmarks, support of supercomputer: Rapid testing
- C++ for defining environments.
- Exact algorithm for smaller benchmarks

# Considerations



- Do you want to eventually do research in DRL, or only learn about it?
- What is the type of problem you are solving?
- Does your research field require you to benchmark you DRL solution, or is there no appropriate benchmark anyhow?
- Preference for coding language?

# Questions?

# The supply chain game



- Conceived for January 2020 ESCF workshop on deep reinforcement learning in logistics.
  - Assembly-to-order W system (commonality & assembly).

# The supply chain game



- Conceived for January 2020 ESCF workshop on deep reinforcement learning in logistics.
  - Assembly-to-order W system (commonality & assembly).
  - Capacitated supply, lost sales, emergency shipments.

# The supply chain game



- Conceived for January 2020 ESCF workshop on deep reinforcement learning in logistics.
  - Assembly-to-order W system (commonality & assembly).
  - Capacitated supply, lost sales, emergency shipments.
- Goal in this course: Illustrate some shortcomings of neural network policies and how to overcome them.

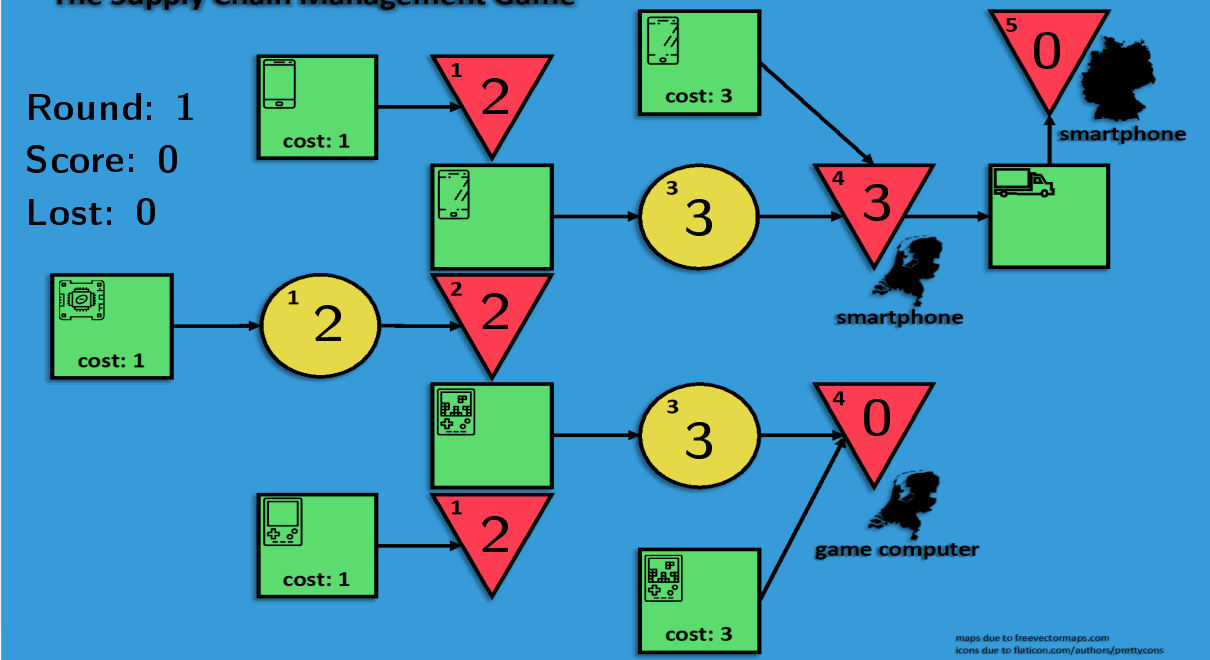
# The supply chain game (2)

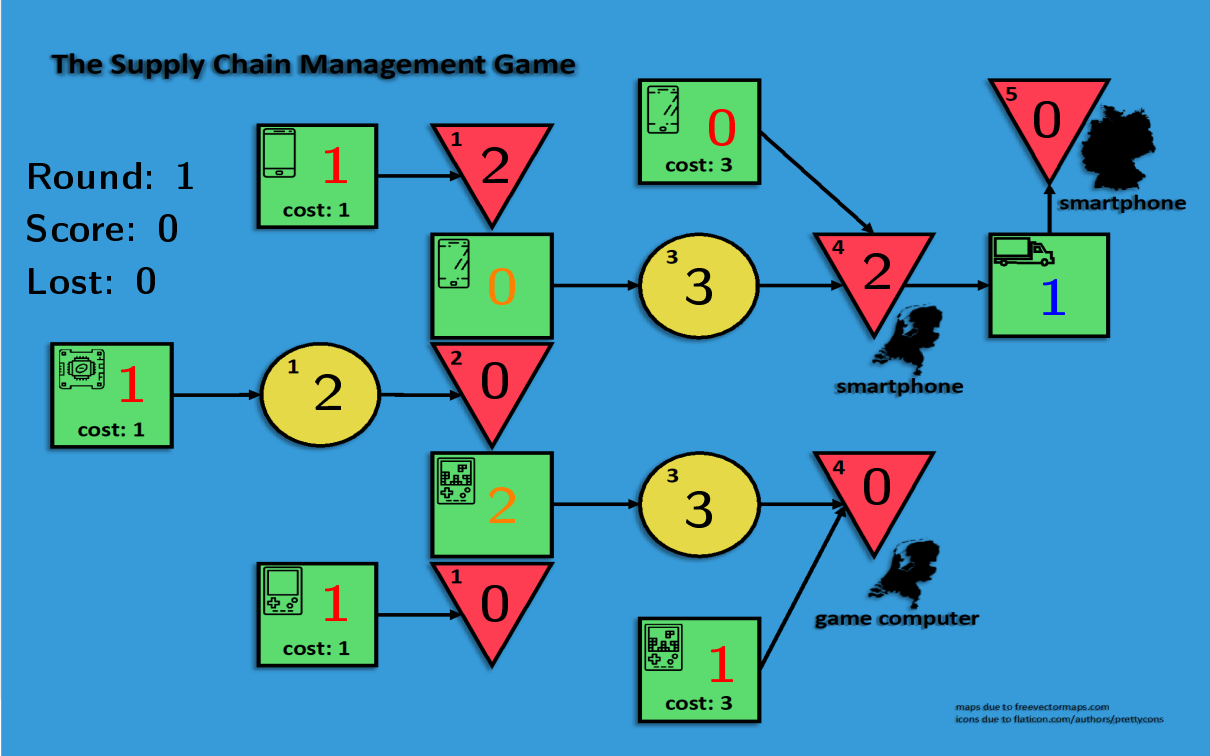


- Played in 10 rounds/periods.
- During each round, we first decide on external orders, assembly, and transshipment.
- Stochastic demand. After each round, demand happens at *one* out of *three* possible demand locations.
- Goal: satisfy as many demands as possible/lose as few demands as possible.

# The Supply Chain Management Game

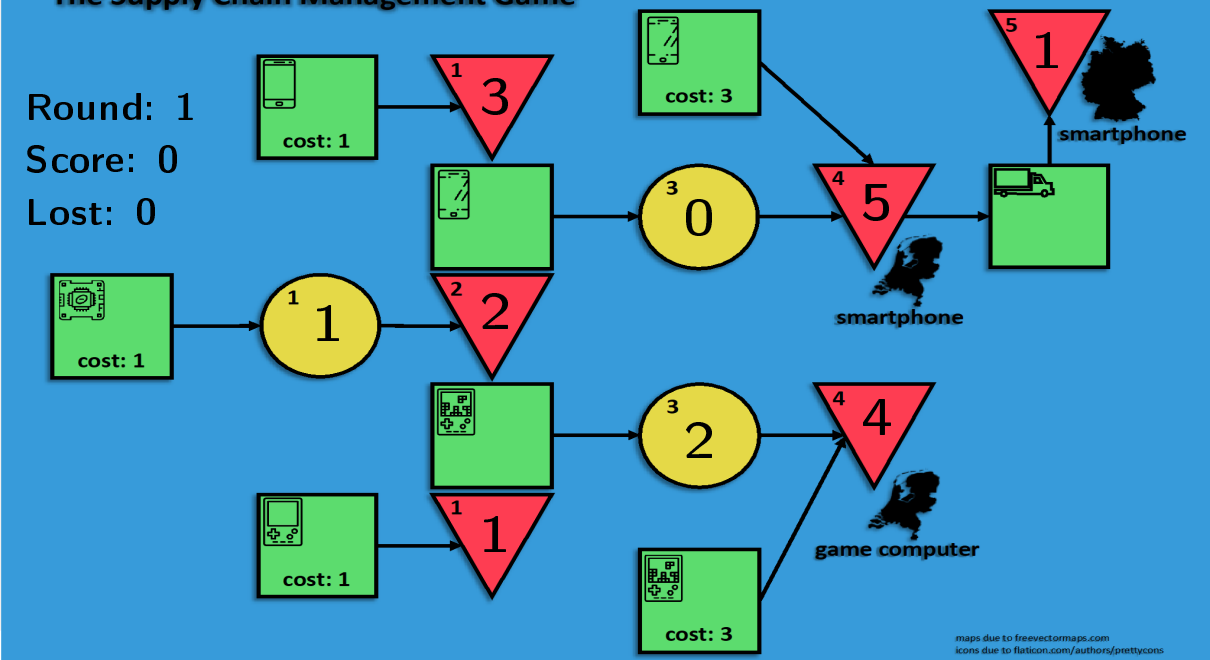
Round: 1
Score: 0
Lost: 0




The Supply Chain Management Game
Round: 1
Score: 0
Lost: 0
1
cost: 1
1
2
0
cost: 3
5
0
smartphone
0
3
4
2
1
smartphone
1
cost: 1
1
2
2
0
2
3
4
0
1
cost: 1
1
0
1
cost: 3
game computer
maps due to freevectormaps.com
icons due to flaticon.com/authors/prettycons

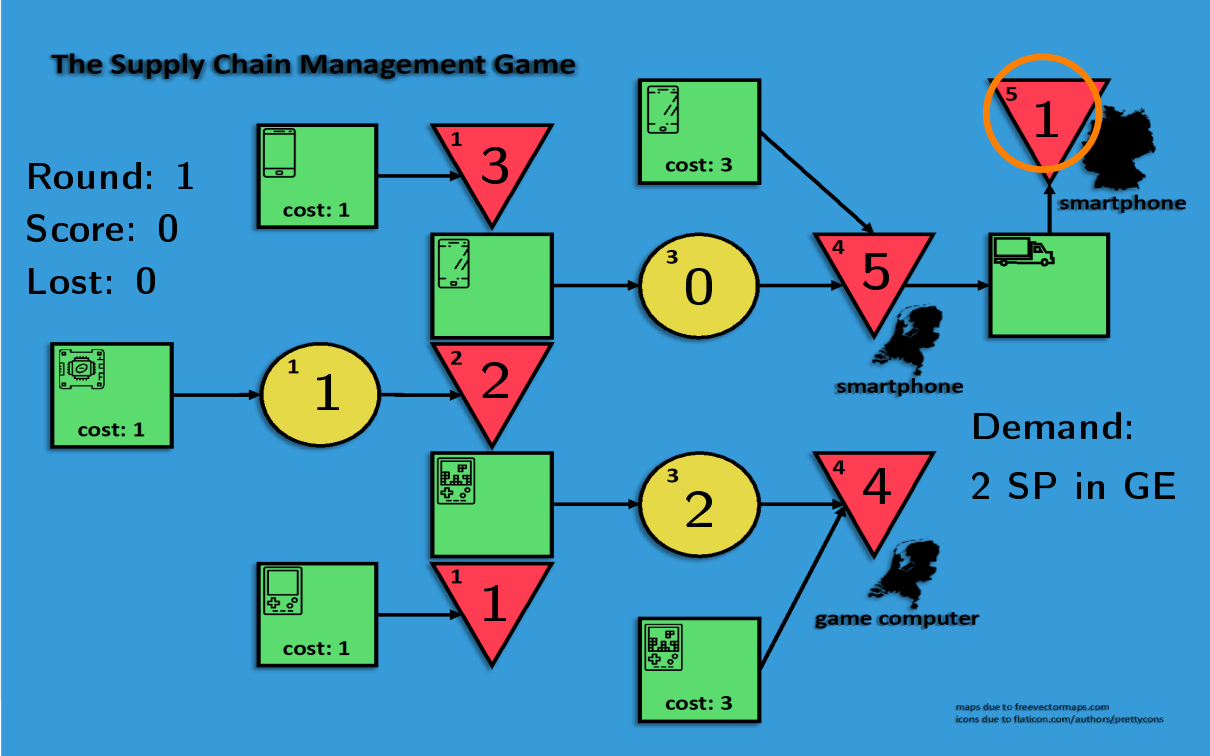
# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0



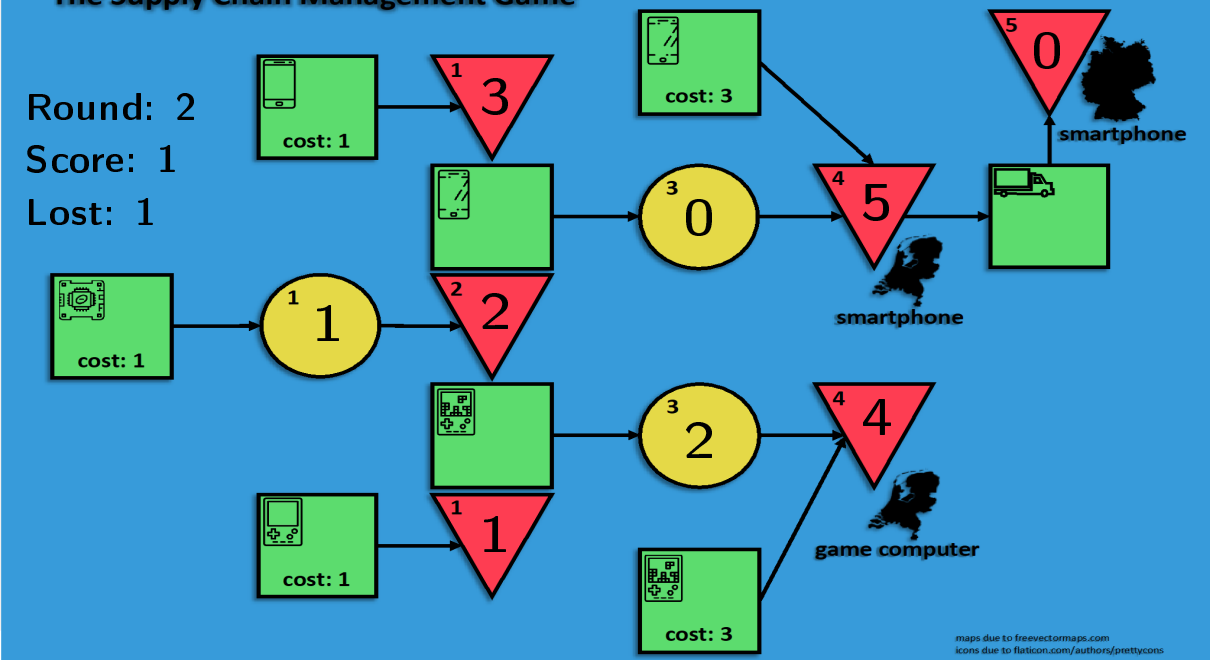
# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0



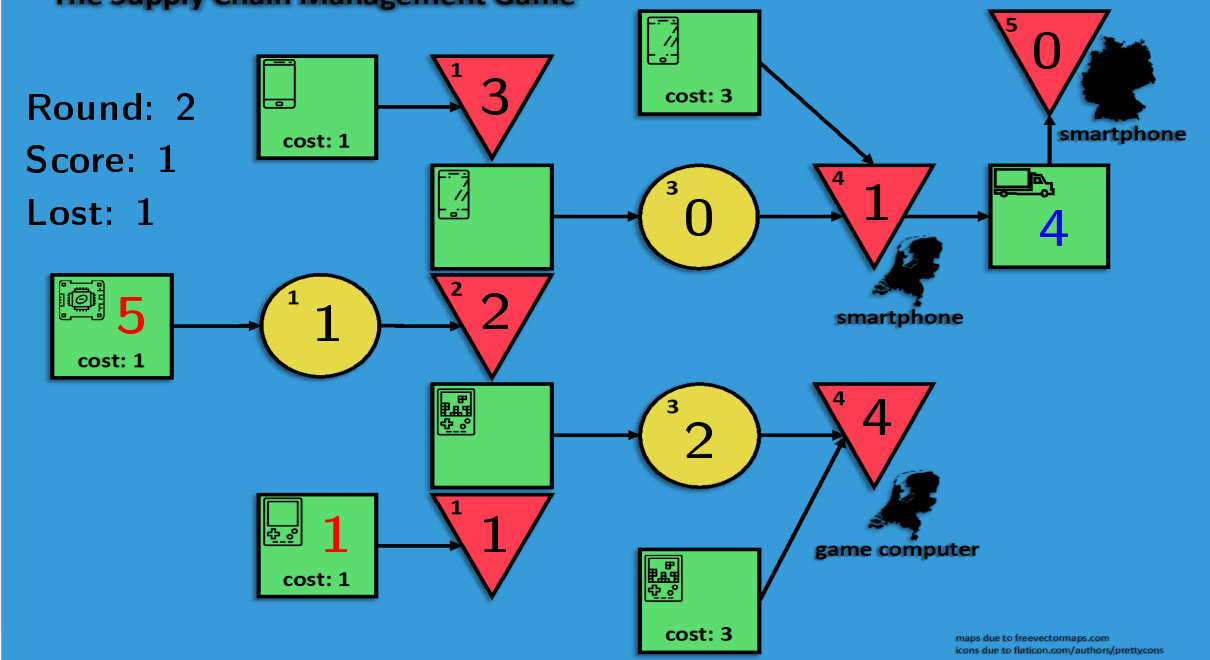
# The Supply Chain Management Game

Round: 2
Score: 1
Lost: 1



# The Supply Chain Management Game

Round: 2
Score: 1
Lost: 1



# A neural network for the SC game



- When approximating the policy as a neural network, what should it look like?
- As *input* we need to have the state, represented as vector in $\mathbb{R}^n$:
  - Inventory for the 9 yellow and red locations, plus 1 dimension for the current round.
- We need one output for each possible action:
  - 51 possibilities for placing external orders (considering capacity of 6).
  - Restricting assembly for each product to be $\leqslant 6$ gives $7 \times 7 = 49$ possibilities.
  - Restricting transshipment quantity to $\leqslant 6$ gives 7 possibilities.
- The action taken each round consists of external orders, assembly, and transshipment: $51 \times 49 \times 7 = 17493$ possible actions.
- So our (naive) neural network needs output dimension in $\mathbb{R}^{17493}$!

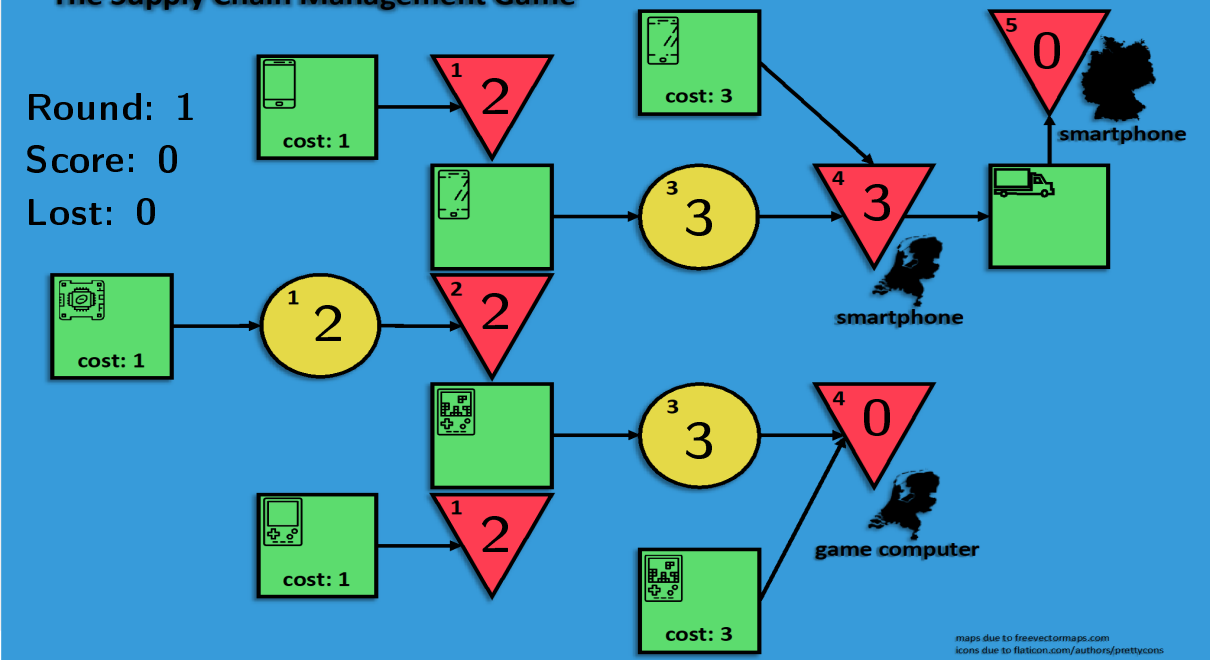
# Representing problems as MDPs



- Considerable freedom in the modeling of (mathematically well-defined) dynamic decision problems as MDPs.
- E.g. we could model the SC game as an MDP by defining the "state" as the inventory at the yellow and red locations at the start of a round, before making a decision.
  - This way, we visit 10 states and take 10 actions before terminating after round 10.

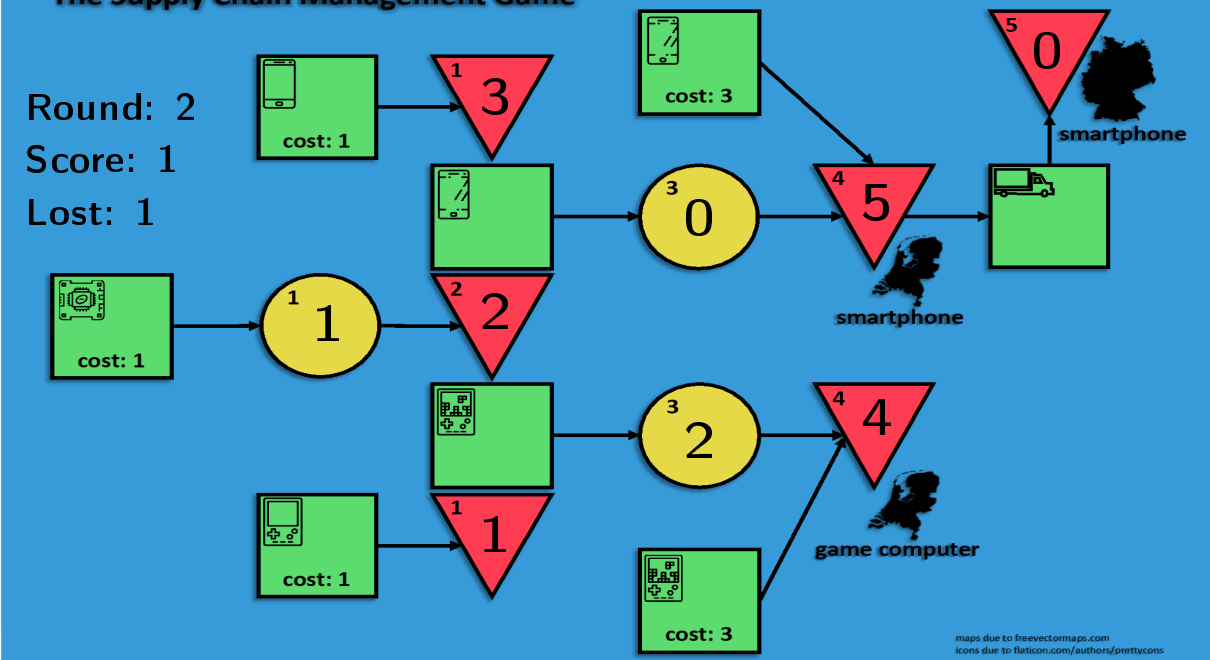
# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0



# The Supply Chain Management Game

Round: 2
Score: 1
Lost: 1



# Representing problems as MDPs



- Considerable freedom in the modeling of (mathematically well-defined) dynamic decision problems as MDPs.
- E.g. we could model the SC game as an MDP by defining the "state" as the inventory at the yellow and red locations at the start of a round, before making a decision.
  - This way, we visit 10 states and take 10 actions before terminating after round 10.

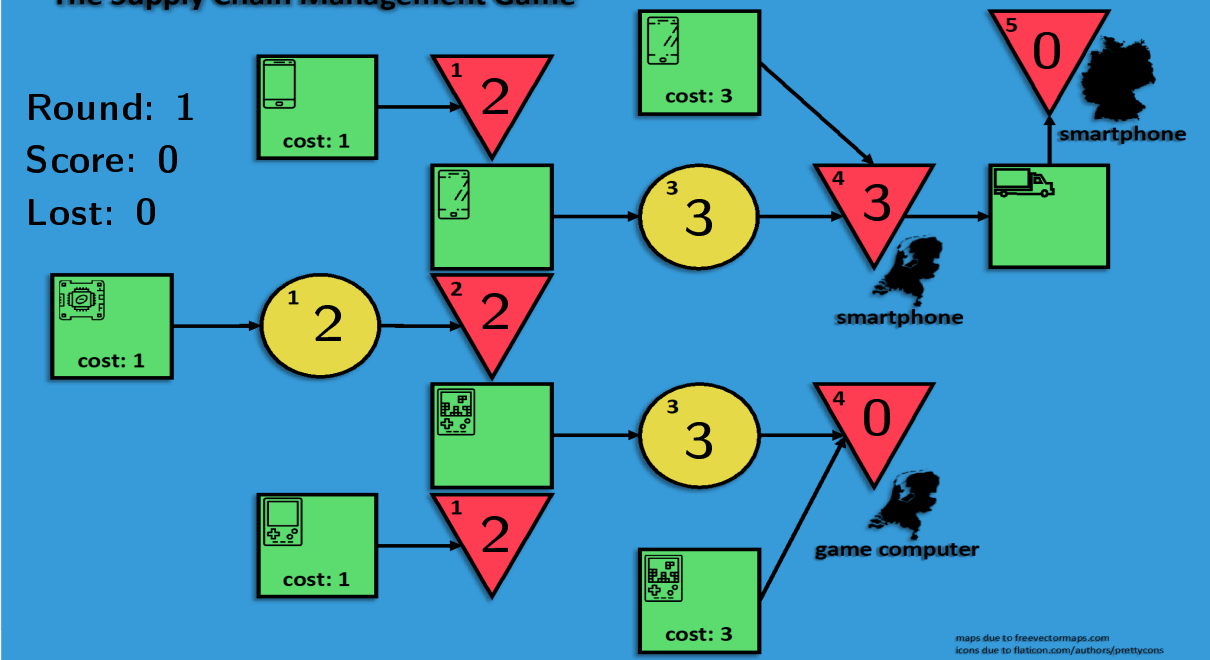
# Representing problems as MDPs



- Considerable freedom in the modeling of (mathematically well-defined) dynamic decision problems as MDPs.
- E.g. we could model the SC game as an MDP by defining the "state" as the inventory at the yellow and red locations at the start of a round, before making a decision.
  - This way, we visit 10 states and take 10 actions before terminating after round 10.
- **Alternatively**, we could define three states for each round: 1) before external orders, 2) before assembly, and 3) before transshipment.
- This way, we visit 30 states and take 30 actions before terminating after round 10.

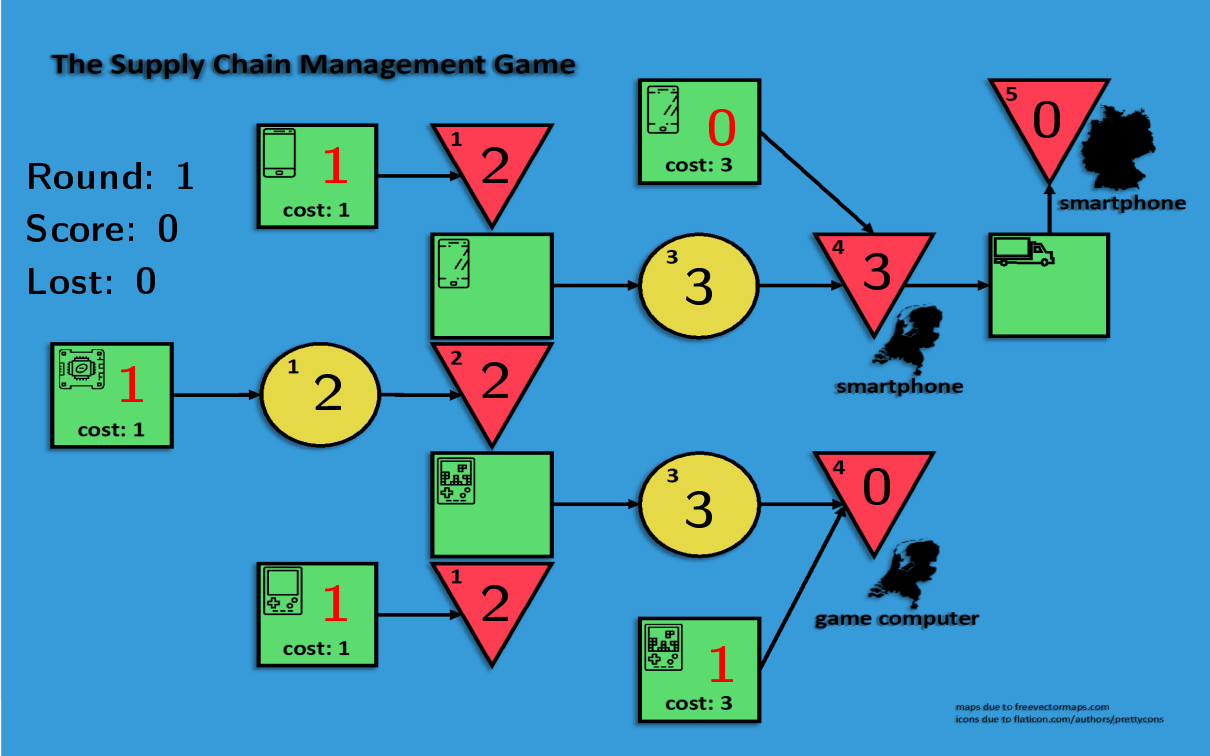
# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0

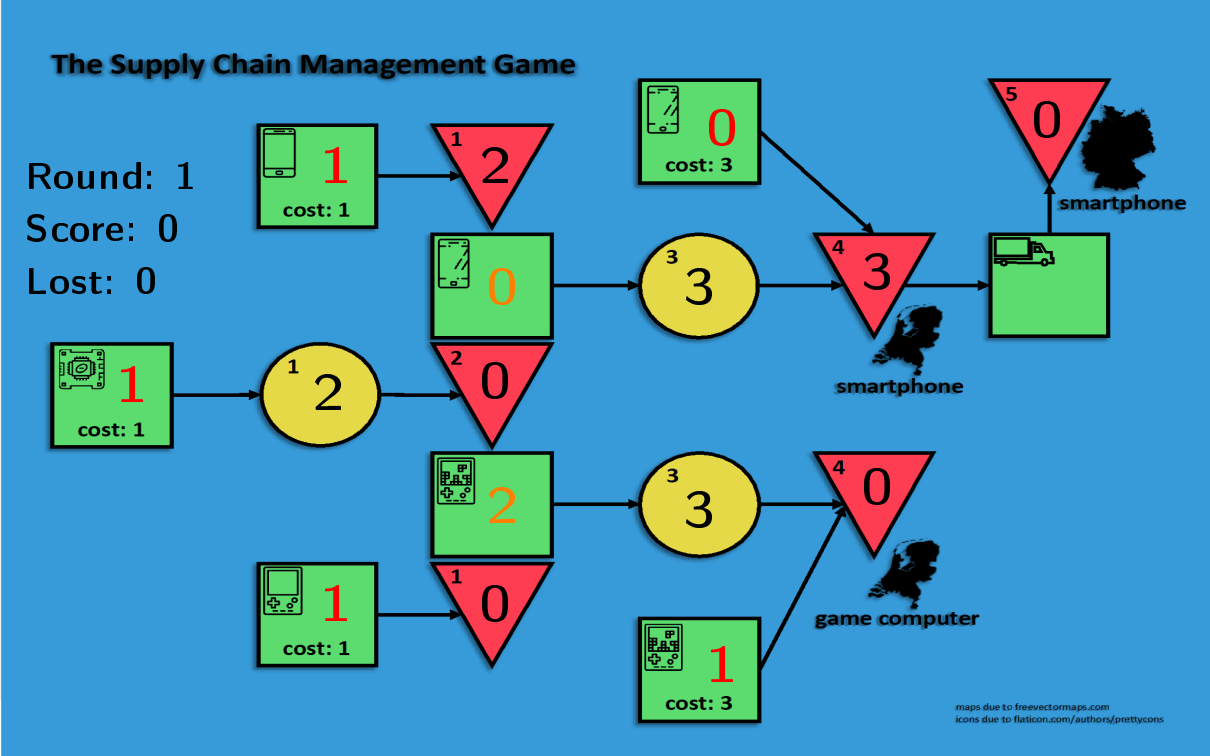


# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0

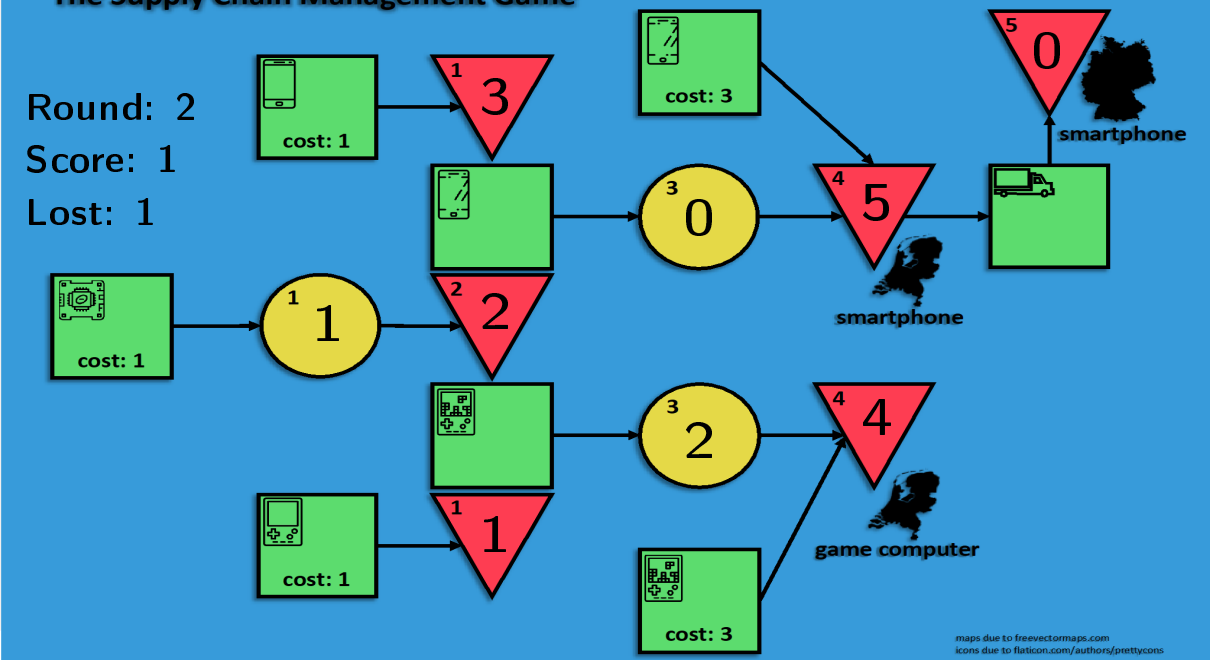


# The Supply Chain Management Game



# The Supply Chain Management Game

Round: 2
Score: 1
Lost: 1



# Representing problems as MDPs



- Considerable freedom in the modeling of (mathematically well-defined) dynamic decision problems as MDPs.
- E.g. we could model the SC game as an MDP by defining the "state" as the inventory at the yellow and red locations at the start of a round, before making a decision.
  - This way, we visit 10 states and take 10 actions before terminating after round 10.
- **Alternatively**, we could define three states for each round: 1) before external orders, 2) before assembly, and 3) before transshipment.
- This way, we visit 30 states and take 30 actions before terminating after round 10.

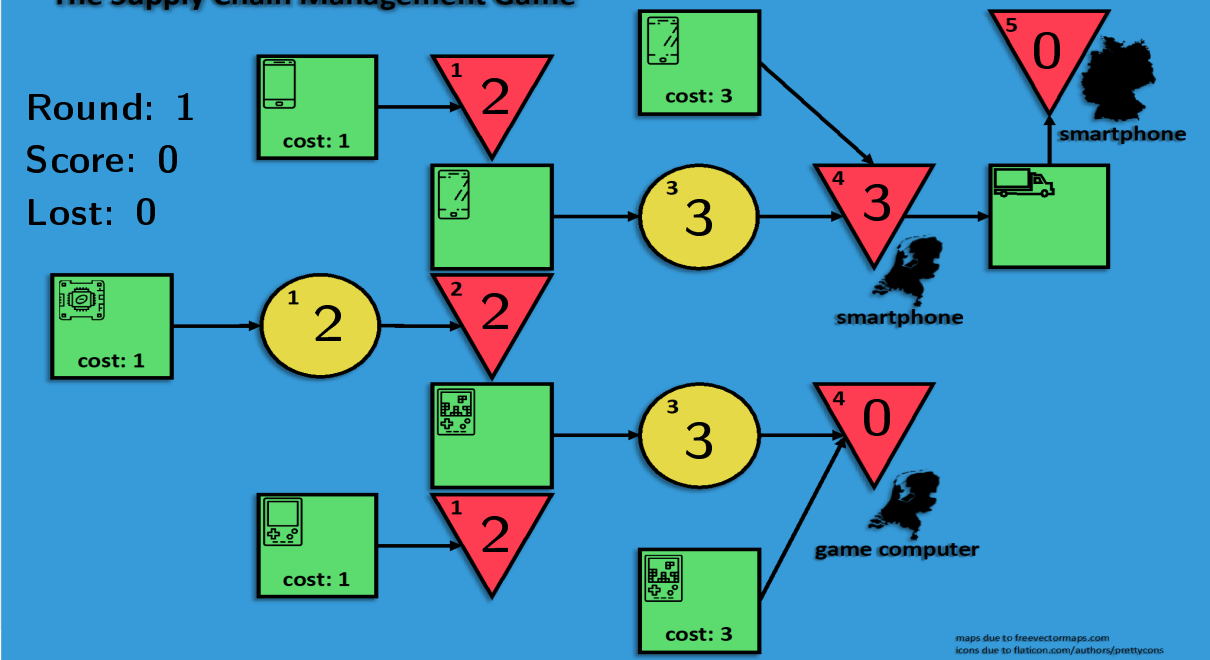
# Representing problems as MDPs



- Considerable freedom in the modeling of (mathematically well-defined) dynamic decision problems as MDPs.
- E.g. we could model the SC game as an MDP by defining the "state" as the inventory at the yellow and red locations at the start of a round, before making a decision.
  - This way, we visit 10 states and take 10 actions before terminating after round 10.
- **Alternatively**, we could define three states for each round: 1) before external orders, 2) before assembly, and 3) before transshipment.
- This way, we visit 30 states and take 30 actions before terminating after round 10.
- To preserve the Markov property we then need to expand the state space.
  - Before deciding upon transshipment, we get information on external orders and assembly.

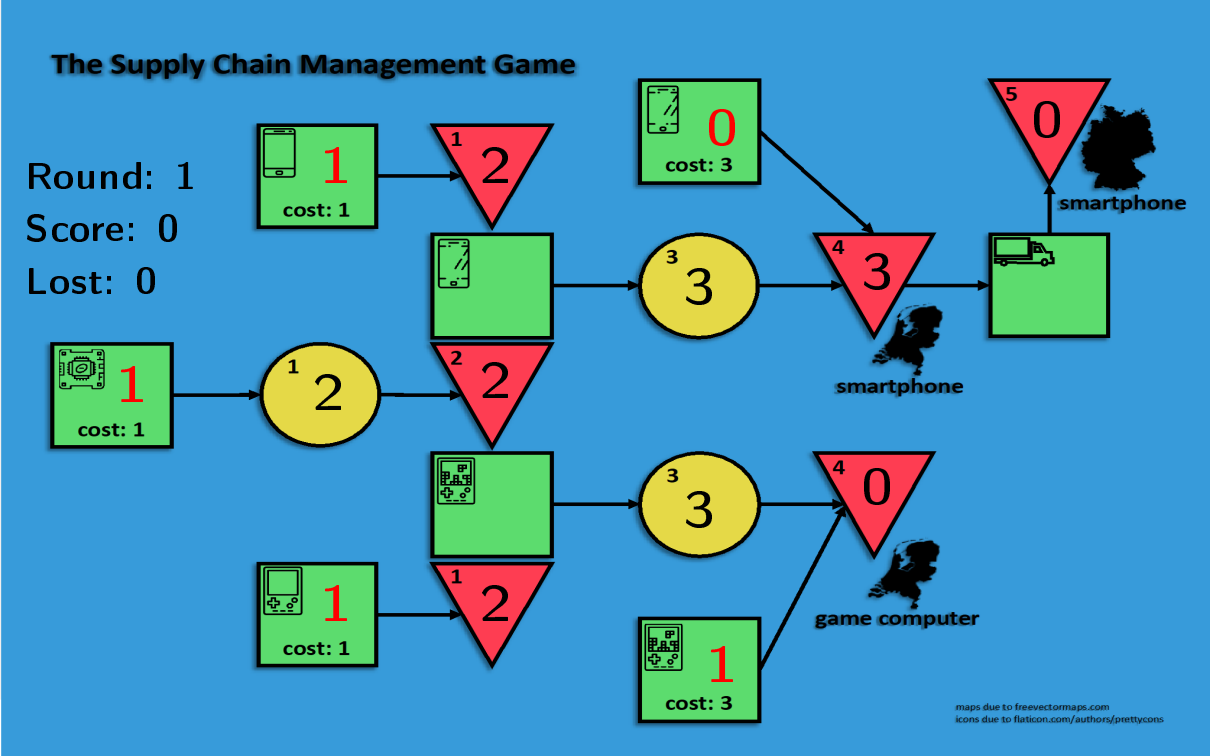
# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0

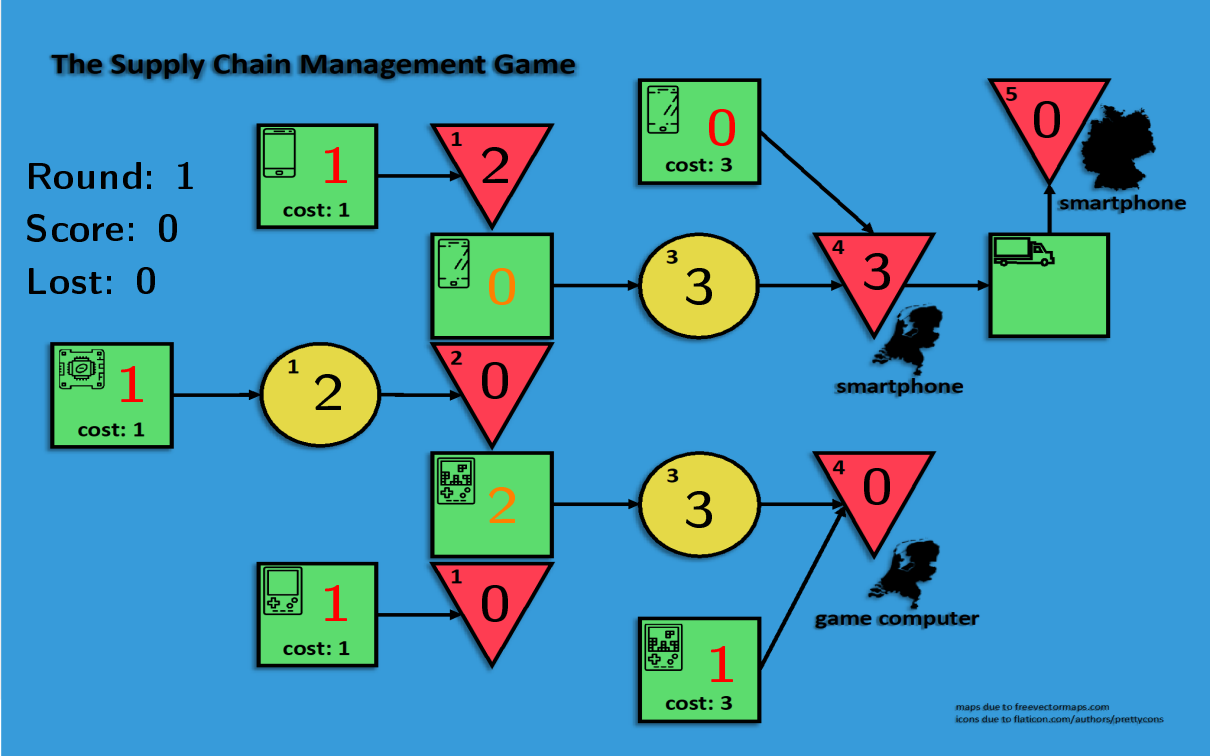


# The Supply Chain Management Game

Round: 1
Score: 0
Lost: 0

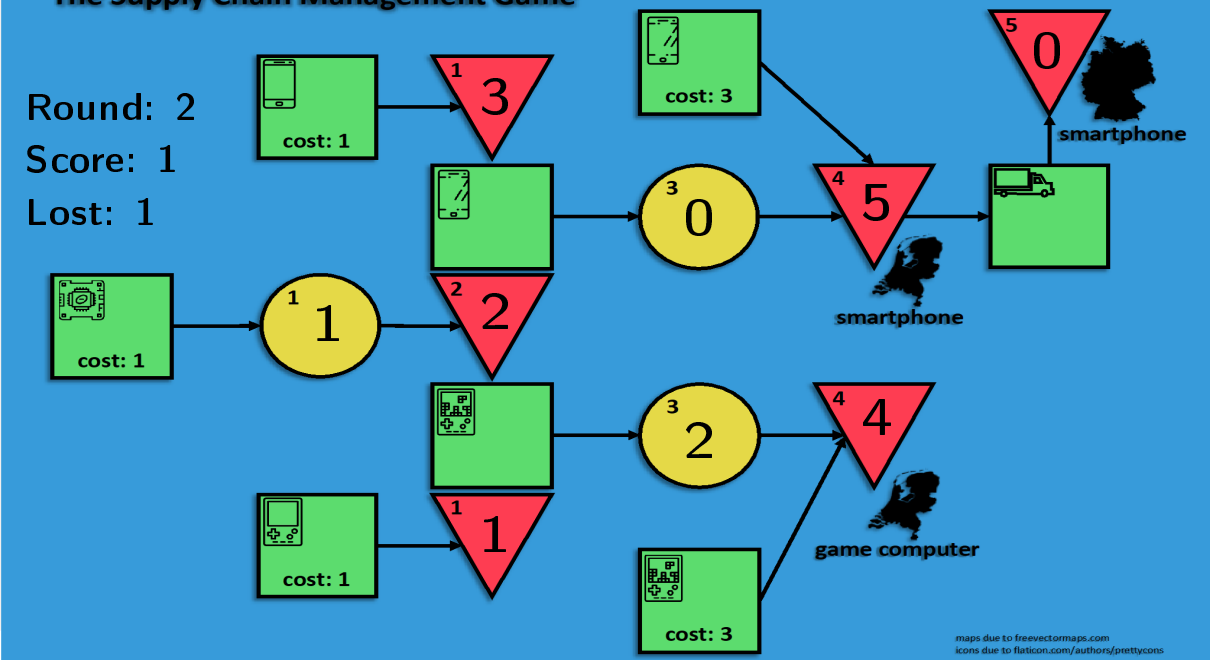


# The Supply Chain Management Game



# The Supply Chain Management Game

Round: 2
Score: 1
Lost: 1



# Neural networks



- If we choose the latter representation, then we avoid the action space explosion (to some extent).
- But what should be the approach for the neural networks:
  1. The actions available are very different for the three types of states.
  2. The state space is a bit different for across the three types of states.
- Two related approaches for dealing with point 1:
  - Three neural networks, one for external orders, one for assembly, and one for transshipment, OR

# Neural networks



- If we choose the latter representation, then we avoid the action space explosion (to some extent).
- But what should be the approach for the neural networks:
  1. The actions available are very different for the three types of states.
  2. The state space is a bit different for across the three types of states.
- Two related approaches for dealing with point 1:
  - Three neural networks, one for external orders, one for assembly, and one for transshipment, OR
  - One neural network to rule them all.

# Neural networks



- If we choose the latter representation, then we avoid the action space explosion (to some extent).
- But what should be the approach for the neural networks:
  1. The actions available are very different for the three types of states.
  2. The state space is a bit different for across the three types of states.
- Two related approaches for dealing with point 1:
  - Three neural networks, one for external orders, one for assembly, and one for transshipment, OR
  - One neural network to rule them all.

# One network for multiple state types



Things to be dealt with:

1. State spaces are a bit different:
   - Solution: Use 16 features for demand locations (including decision locations). Set features that are not applicable for state to 0.

# One network for multiple state types



Things to be dealt with:

1. State spaces are a bit different:
   - Solution: Use 16 features for demand locations (including decision locations). Set features that are not applicable for state to 0.
2. How to translate the output of the network to actions?
   - Remember there are 51, 49, and 7 actions for the three possible states types.
   - We design the neural network with output dimension $m = 51 + 49 + 7 = 107$.
     - First 51 outputs correspond to external orders, next 49 to assembly, etc.
   - For inferencing, we take the soft max/arg max only over the outputs that are applicable for the state we are inferencing for.
   - I.e. when in a state that corresponds to placement of assembly orders, actions corresponding to external orders and transshipment are not allowed.

# Questions?