Solving a Product Delivery Problem with Reinforcement Learning and Deep Neural Networks

Dani Salgado

Advisor: Toni Lozano

Master's thesis

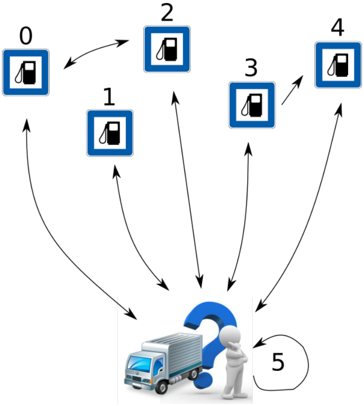
Universitat Autònoma de Barcelona

GRUPO AIA

Outline

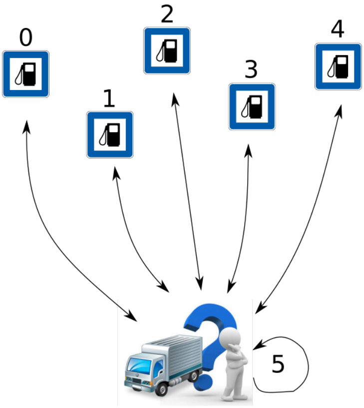
Problem statement

- Imagine we own a chain of stores specialized in a particular product, and we have to pay to a transport company to bring our product from a depot to the shops.
- The goal is to minimize the amount of money to pay to the transport company, but ensuring that there is always gas available for costumers (among other possible constraints).



Initial assumptions

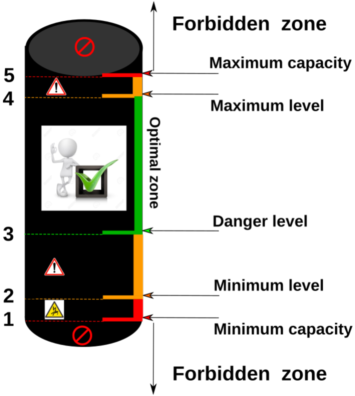
1. We have a system of n shops and k trucks.
2. A truck can go to at most **one** shop everyday.
3. Trucks leave the depot fully loaded and return completely empty.
4. The price to pay to the transport company is given by some **cost function** J_{costs} which may depend on the distance travelled by the trucks, the amount of product delivered, if it is a holiday or not, etc.



Levels of stock (modelling wellness of shops)

We assume to have a criteria to say if the current level of stock in a given shop is “good or bad”.

- **Maximum (resp. minimum) capacity**: it is physically impossible or very dangerous to have a stock **above (resp. below)** this level.
- **Maximum level**: desired maximum stock.
- **Danger level**: expected consumption in the next 36 hours.
- **Minimum level**: expected consumption in the next 12 hours.



Final goal

Our product delivery problem becomes an **optimization problem** that needs to balance transport company costs (J_{costs}) and the levels of stock of the shops (we need some "J_{levels}").

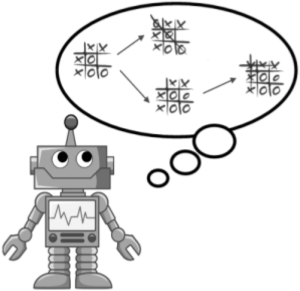
Thus, there is a **trade-off between minimizing transport company costs and maximizing the *wellness* of the shops in terms of stock**.

Outline

Reinforcement learning: states and actions

Consider the picture on the right,

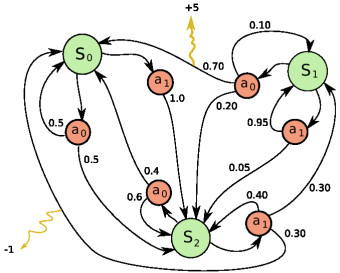
- We assume to have an **agent** (a robot, an algorithm) living in an **environment** (e.g. a tic tac toe board),
- a set of **states** $\mathcal{S}$ (e.g. the possible configurations of 'O' and 'X'),
- and a set of **actions** $\mathcal{A}$ (e.g. move to one of the empty positions in the board).



MDPs, the mathematical model in Reinforcement Learning

Definition

A **Markov decision process** (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, T, R)$ in which $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, T a transition probability function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and R a reward function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. One says that the pair T, R define the *model* of the MDP.



Control of a MDP: agent's policy

- To fix ideas we define a **deterministic policy** π as

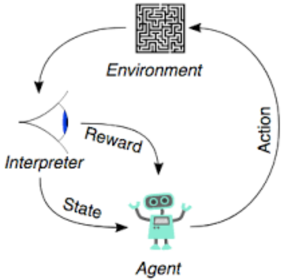
$$\begin{aligned} \pi : \mathcal{S} &\longrightarrow \mathcal{A} \\ s &\longmapsto \pi(s) = a \end{aligned}$$

- In general, one has a stochastic policy

$$\begin{aligned} \pi : \mathcal{S} \times \mathcal{A} &\longrightarrow [0, 1] \\ (s, a) &\longmapsto P(a|s) \end{aligned}$$

  (probability of taking action a given that we are in state s).

- The goal is to **learn an optimal policy** π^* that takes the actions that maximize rewards.

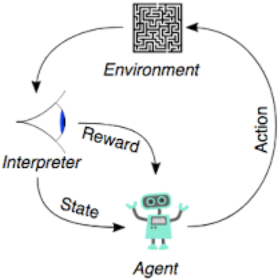


Controlling the environment using a policy π

A policy π can be used to make evolve a MDP system in the following way:

- Starting from an initial state $s_0 \in \mathcal{S}$, the next action the agent will do is taken as $a_0 = \pi(s_0)$.
- After the action is performed by the agent, a transition is made from s_0 to some state s_1, with probability $T(s_0, a, s_1)$ and a obtained reward $r_0 = R(s_0, a_0, s_1)$.
- By iterating this process, one obtains a sequence $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$ of state-action-reward triples.

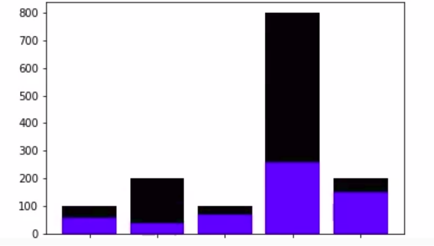
We consider **episodic** tasks so that the sequence of state-action-reward triples ends in a finite number of iterations τ. To fix ideas, we consider $\tau = 30$ (days).

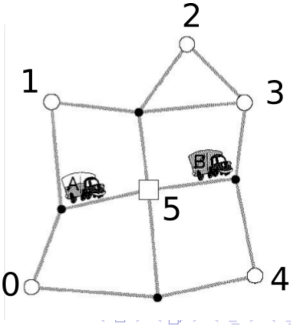


Outline

States and Actions

- n shops, k trucks, (remind assumptions), $\tau = 30$ (each step t of an episode will correspond to one day)
- **States**: $s = (c_1, ..., c_n)$, c_i the stock of shop i.
- **Actions**: $a = (p'_1, ..., p'_k)$, p'_i the position of truck i.





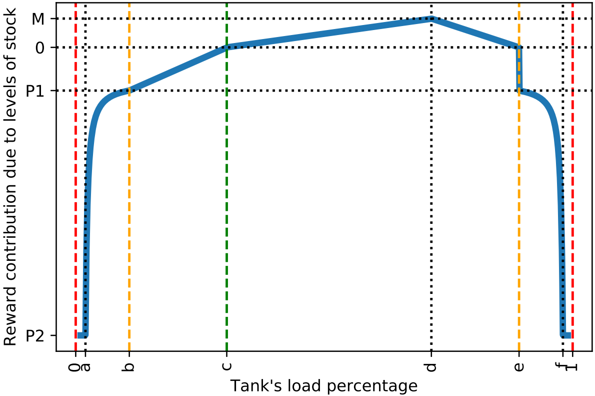
Function of rewards

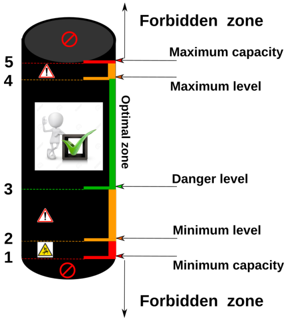
$$R(s, a, s') = \mu_1 J_{\text{costs}} + \mu_2 J_{\text{levels}} + \sum_j \mu_{3,j} J_{\text{extra},j}, \qquad \mu_i \geq 0,$$

Three contributions:

- J_{costs}: **economical costs** such as transport distances, amount of product unloaded, holidays,...
- J_{levels}: **levels of stock** of each shop,
- Additional terms such as $J_{\text{extra},1}$, a cost for **penalizing a truck that goes to some shop but can't deliver** its product (because the shop is too full).

J_{levels} for each shop





Summary: Reinforcement learning and Model

Problem: how, when and where the trucks have to be sent to the shops in order to maximize rewards R: find optimal policy π^*.

- We have a MDP model for $\mathcal{S}, \mathcal{A}$ and R,
  - n shops, k trucks, (and some assumptions)
  - **States**: $s = (c_1, ..., c_n) \in \mathcal{S}$,
  - **Actions**: $a = (p'_1, ..., p'_k) \in \mathcal{A}$.
  - **Rewards** function:

$$R(s, a, s') = \mu_1 J_{\text{costs}} + \mu_2 J_{\text{levels}} + \mu_{3,1} J_{\text{extra},1}, \qquad \mu_i \geq 0,$$

- but we don't know the transition operator T. Thus, we need RL algorithms that do not need prior knowledge of T: e.g., **Q-learning**.

Outline

The Q values: a way to quantify goodness of state-action pairs

Definition

The **state-action value of** s**,**a **under policy** π, denoted $Q^\pi(s,a)$ is the expected return when starting in state s, taking action a and thereafter following π:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\tau} \gamma^k R_{t+k} | S_t = s, A_t = a\right]$$

where $\tau < \infty$ if the task is episodic, and $\tau = \infty$ if it is continuing.

Optimal Q-values and an optimal policy

Definition

A policy π^* is said to be **optimal** if it is such that $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$ and all policies π. $Q^* := Q^{\pi^*}$ is called the optimal value function.

Assume we know $Q^*(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$, or an algorithm able to estimate them. Then, one can greedily select an optimal action using the greedy Q-policy π_Q defined as

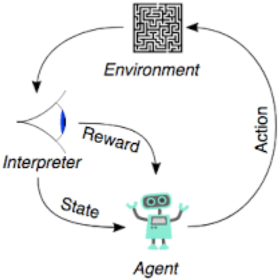
$$\boxed{\pi_Q(s) = \arg\max_{a \in \mathcal{A}(s)} Q^*(s,a), \qquad \forall s \in \mathcal{S},}$$

and π_Q **is an optimal policy** according to the definition above.

Learning by experience

To learn the optimal Q values one simulates different episodes E_j using some exploration-exploitation policy π for selecting actions:

$$
\begin{aligned}
E_j &= (s_0^j, \pi(s_0^j), r_0^j, s_1^j, \pi(s_1^j), r_1^j, \ldots, s_{\tau-1}^j, \pi(s_{\tau-1}^j), r_{\tau-1}^j, s_\tau^j) \\
&= (s_0^j, a_0^j, r_0^j, s_1^j, a_1^j, r_1^j, \ldots, s_{\tau-1}^j, a_{\tau-1}^j, r_{\tau-1}^j, s_\tau^j)
\end{aligned}
$$



One uses **the ε-greedy policy**:

$$
\pi_\varepsilon(s) = \begin{cases} \text{random action from } \mathcal{A}(s) & \text{if } p < \varepsilon \\ \pi_Q(s) = \arg\max_{a \in \mathcal{A}(s)} Q(s,a) & \text{otherwise,} \end{cases}
$$

where $p \in [0,1]$ is a uniform random number drawn at each time step (of each episode).

Outline

Model used in the Q-learning simulations

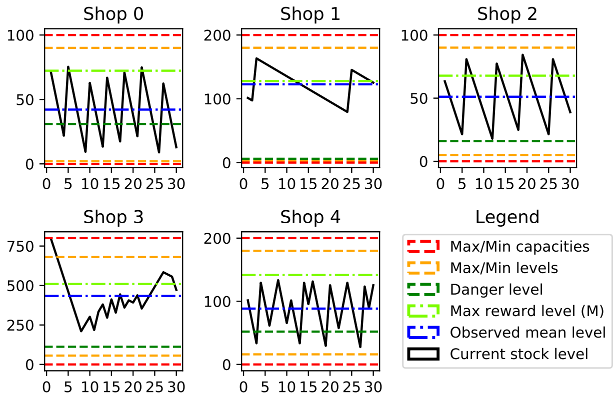
- Case $n = 5$ shops, $k = 2$ trucks.
  - Shops of capacity 100, 200, 100, 800, 200.
  - Trucks of capacity 70, 130.
- We discretize **states**: $s = (c_0, \dots, c_{n-1})$ in 4 subintervals for each shop,
- **Actions**: $a = (p'_0, \dots, p'_{k-1})$.
- Q-learning simulations of 500.000 episodes of length $\tau = 30$ (days)
- The learnt $Q(s, a)$-values are a tabular function (one row per state and one column per action).

Simulations

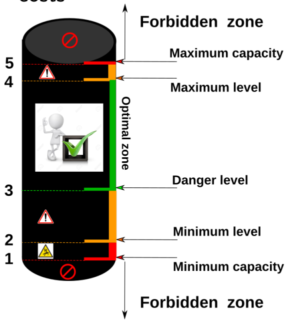
$$R(s,a,s') = \mu_1 J_{\text{costs}} + \mu_2 (J_{\text{levels}} + J_{\text{extra},1}), \qquad \mu_1, \mu_2 \geq 0.$$

1) Deterministic consumption rates without transport costs: $\mu_1 = 0$.
2) Deterministic consumption rates WITH transport costs: $\mu_1 \neq 0$.
3) Stochastic consumption rates ($\pm 10\%$) without transport costs: $\mu_1 = 0$.
4) Stochastic consumption rates ($\pm 10\%$) WITH transport costs: $\mu_1 \neq 0$.

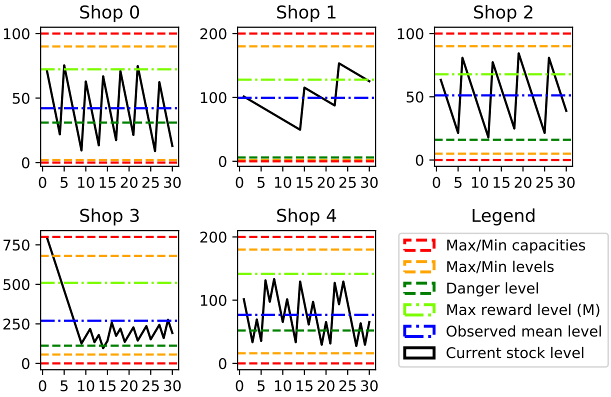
1) Deterministic without costs, $\mu_1 = 0$



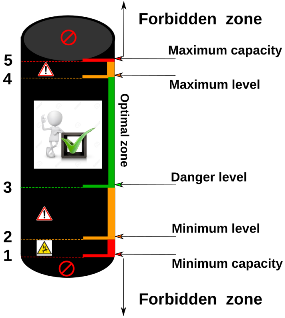
43 trucks sent, $J_{\text{costs}} = 29407$



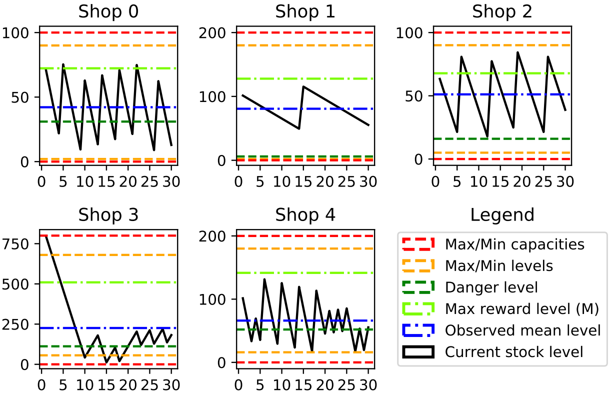
2.1) Deterministic with costs, $\mu_1 = 10^{-6}$



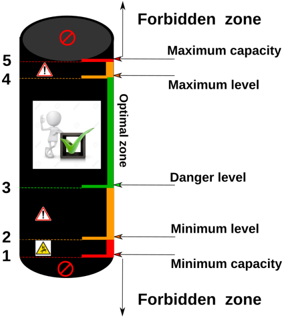
40 trucks sent, $J_{\text{costs}} = 27717$



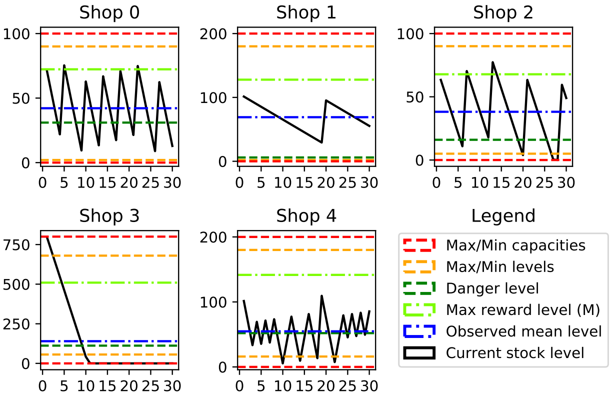
2.2) Deterministic with costs, $\mu_1 = 10^{-4}$



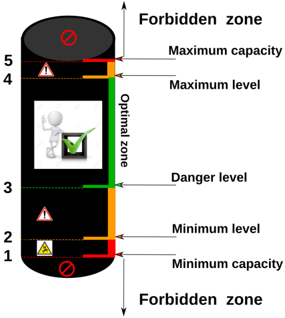
37 trucks sent, $J_{\text{costs}} = 26982$



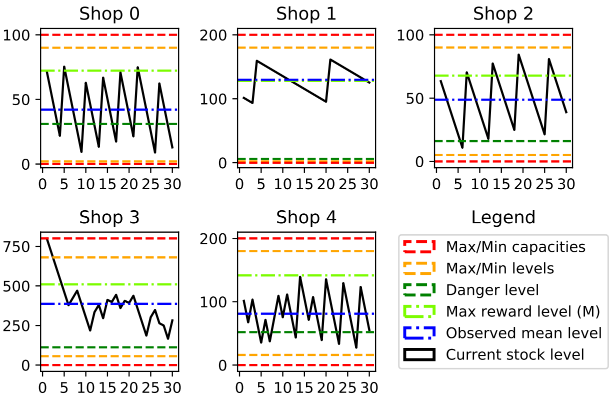
2.3) Deterministic with costs, $\mu_1 = 10^{-3}$



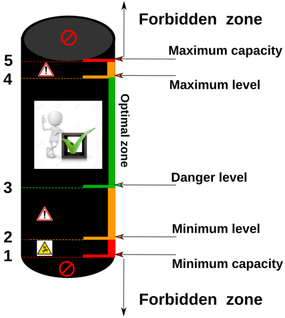
25 trucks sent, $J_{\text{costs}} = 11530$



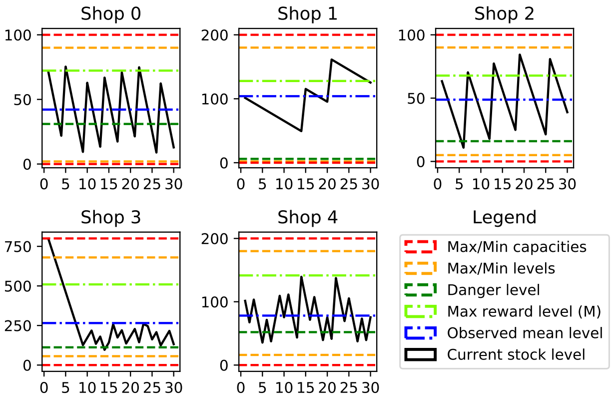
3) Stochastic without costs, $\mu_1 = 0$ (10% noise)



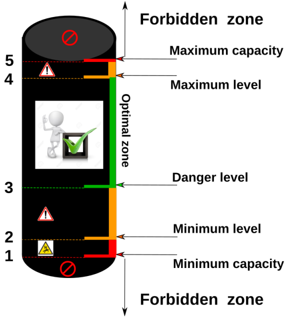
42 trucks sent, Shops 3 & 4: 26 small and 16 big, $J_{\text{costs}} = 28361$



4) Stochastic with costs, $\mu_1 = 10^{-6}$ (10% noise)



41 trucks sent,
Shops 3 & 4: 26 small and 15 big,
$J_{\text{costs}} = 27315$



Conclusions

Q-learning could be very **effective** for a small number of shops and trucks, **but the number of state-action pairs explodes**:

$$|\mathcal{S}| = 4^n, \text{ if we discretize each shop in four parts}$$
$$|\mathcal{S} \times \mathcal{A}| = 4^n \times (n+1)^k \approx 36800 \text{ for n=5, k=2}$$

but

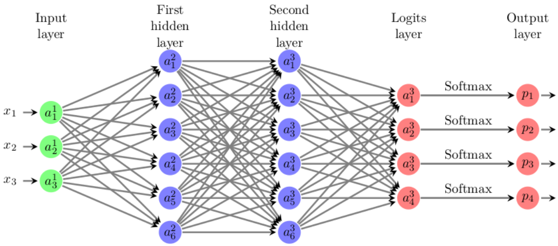
$$|\mathcal{S} \times \mathcal{A}| = 4^n \times (n+1)^k \approx 10^{16} \text{ for n=20, k=3}$$

Even so, discretizing in only 4 parts for each shop we are losing a lot of information about the system.

Outline

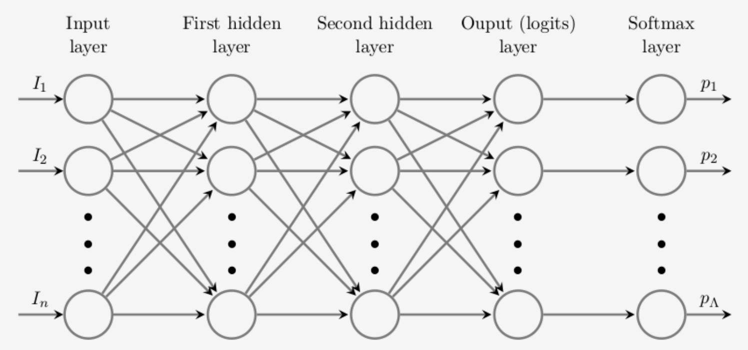
What is next?: Deep Reinforcement Learning

- Make the problem scalable to "arbitrary" number of shops and trucks.
- Use Deep Neural Networks for Reinforcement Learning: **Deep Policy gradient algorithm** (our approach).
- A DNN will be a parametrized policy π_θ, and the goal is to optimize this policy by updating the parameters θ (the weights and biases).



DNN architecture

The inputs of the network are the states $s = (c_1, ..., c_n) \subset \mathbb{R}^n$, where c_i are the "current" stocks of each shop; $I_j = c_j \ \forall j$. There are $\Lambda = (n+1)^k$ output neurons, one for each possible action. We have: $p_i \equiv \pi_\theta(a_i|s)$.



Outline

How do we train our DNN?: Policy Gradient Theorem

The cost function to consider is the total discounted reward obtained in every episode,

$$J_\theta = \mathbb{E}_{\pi_\theta}\left[R_0^\gamma\right] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\tau-1} \gamma^t R_t\right].$$

Theorem (Policy Gradient)

Assuming an episodic MDP, for any differentiable policy $\pi_\theta(a|s)$ and the policy loss function J_θ defined in equation (35), the policy gradient is

$$\nabla_\theta J_\theta = \mathbb{E}_{\pi_\theta}\left[\left(\sum_{t=0}^{\tau-1} \gamma^t R_t\right)\left(\sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(A_t|S_t)\right)\middle| S_0 = s, A_0 = a\right].$$

Policy Gradient Theorem Estimation

PG theorem gives us a way to estimate the gradient of the loss function by sampling some number N of episodes $E_j = \{s_0^j, a_0^j, r_0^j, s_1^j, a_1^j, r_1^j, ..., s_{\tau-1}^j, a_{\tau-1}^j, r_{\tau-1}^j, s_\tau^j\}$ - for $j = 1, ..., N$ - as follows:

$$\nabla_\theta J_\theta \approx \frac{1}{N}\sum_{j=1}^{N}\left[\left(\sum_{t=0}^{\tau-1}\gamma^t r_t^j\right)\left(\sum_{t=0}^{\tau-1}\nabla_\theta \log \pi_\theta(a_t^j|s_t^j)\right)\right]$$

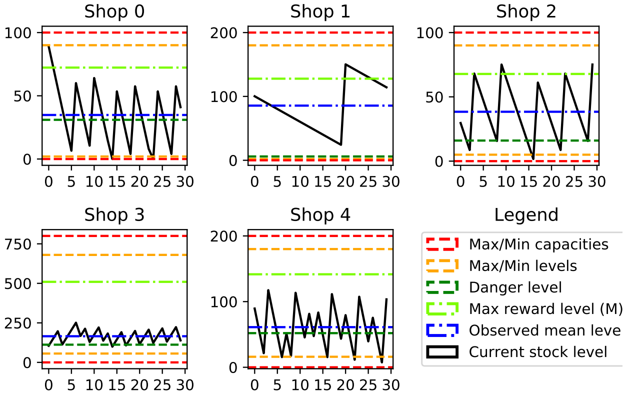
With this we can perform (for instance) a Gradient Ascent step to train the parameters of our DNN:

$$\theta^{k+1} \leftarrow \theta^k + \alpha \nabla_\theta J_\theta \Big|_{\theta=\theta^k},$$

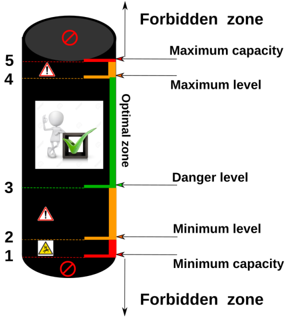
where $\alpha > 0$ is the learning rate.

Outline

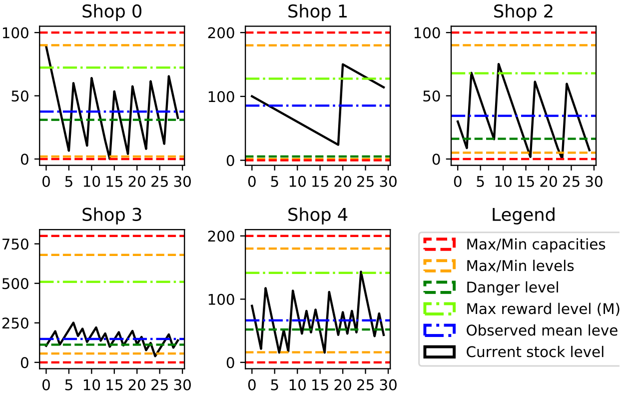
1) Deterministic without costs, $\mu_1 = 0$



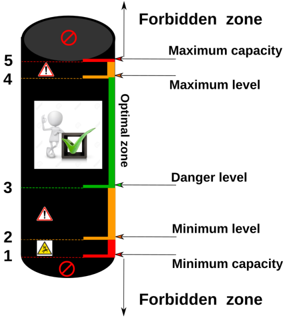
42 trucks sent, $J_{\text{costs}} = 33658$.



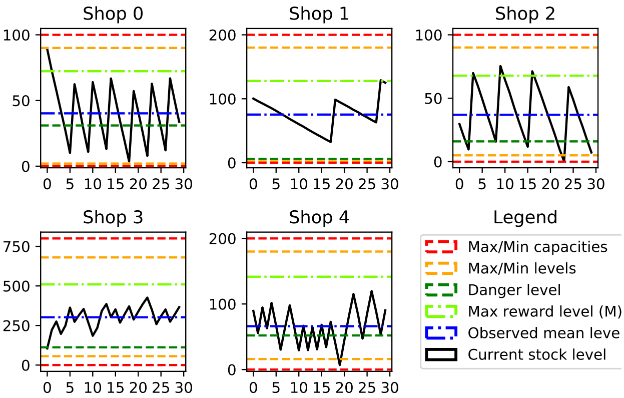
2.1) Deterministic with costs, $\mu_1 = 10^{-6}$



42 trucks sent, $J_{\text{costs}} = 33175$.

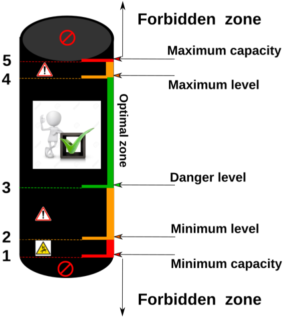


3) Stochastic without costs, $\mu_1 = 0$ (10% noise)

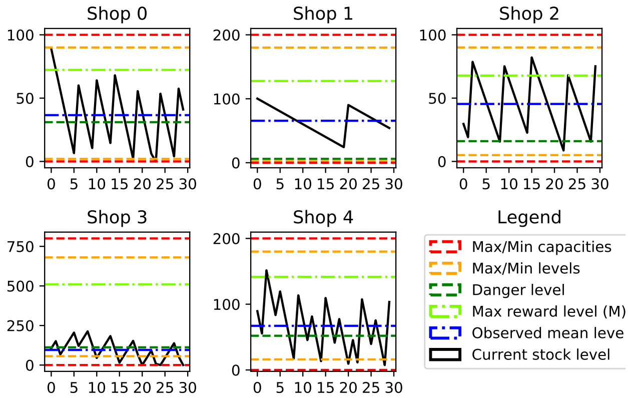


49 trucks sent, Shops 3 & 4: 30 small and 19 big,

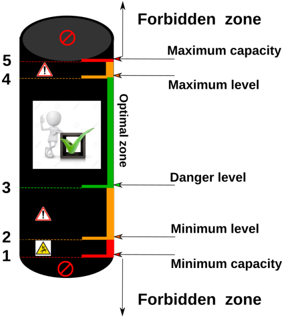
$J_{\text{costs}} = 34212$



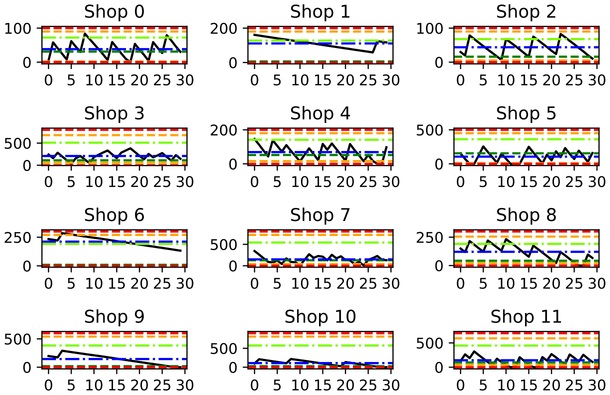
4) Stochastic with costs, $\mu_1 = 10^{-3}$ (10% noise)



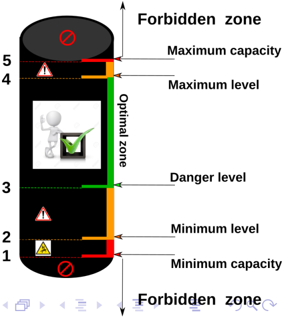
40 trucks sent, Shops 3 & 4: 17 small and 23 big, $J_{\text{costs}} = 31073$



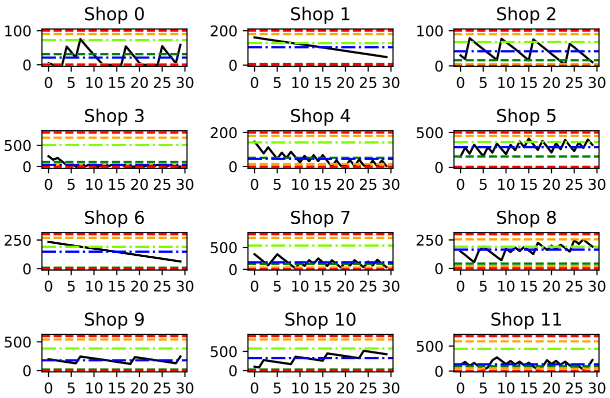
5) Stochastic without costs, $\mu_1 = 0$ (10% noise), $n = 12, k = 3$



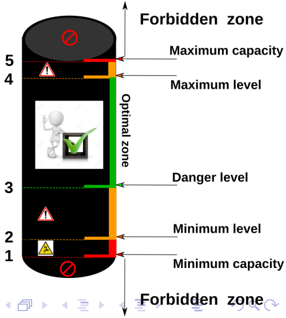
80 trucks sent, 30 small, 30 medium, and 20 big,



6) Stochastic with costs, $\mu_1 = 10^{-6}$ (10% noise), $n = 12, k = 3$



78 trucks sent, 30 small, 30 medium, and 18 big



Outline

Classical v.s. Deep Reinforcement Learning (I)

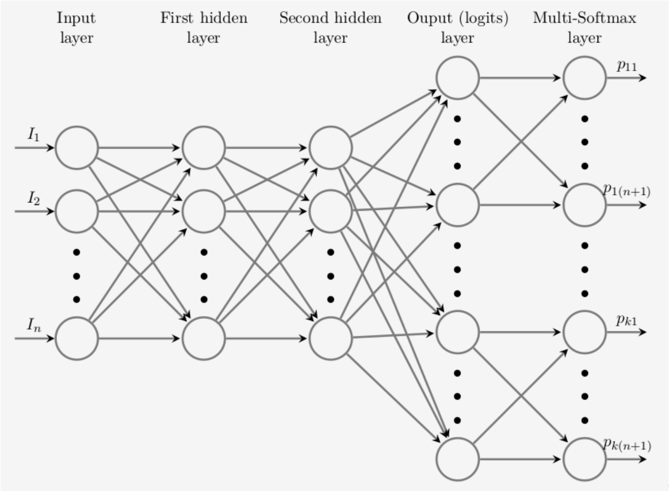
- The **scalability of Q-learning** in terms of the number of shops n and trucks k **is very bad**.

  If d is the number of parts in which we discretize the level of stock of shops, then, the dictionary where we store the Q-values for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, needs to have about $d^n \times (n+1)^k$ keys, a value that grows exponentially with the number of trucks and shops.
- On the contrary, in the PG approach, the dimensionality of $\mathcal{S}$ is not important, since there is no need to discretize the levels of stock of each shop. We just use the exact level of stock of each shop as input of a DNN. In this case **what limits scalability is the total number of possible actions**, since it is equal to the number of neurons in the output layer. The problem is that an order of 10^4 output neurons or more, would be impractical to train in a plausible amount of time.

Classical v.s. Deep Reinforcement Learning (II)

- That is the reason for which we did not consider more than 12 shops and 3 trucks:
  - For $n = 12, k = 3$: $|\mathcal{A}| = 2197 \sim 10^3$,
  - but for $n = 12, k = 4$: $|\mathcal{A}| = 28561 \sim 10^4$.
- Since the scalability is now limited by the number of output neurons $\Lambda = (n+1)^k$, we propose a possible alternative of the current PG approach that would alleviate the scalability from $\Lambda = (n+1)^k$ to $\Lambda = (n+1) \cdot k$, i.e., from exponential to linear scalability on n and k.

More scalable "Deep Policy Gradient" for product delivery



Thanks for your attention! Any questions?

- **Master's thesis, notebooks and figures**:

  `https://github.com/dsalgador/master-thesis`

- **Open AI gym environment for our product delivery problem**:

  `https://github.com/dsalgador/gym-pdsystem`