Ben’s Gelateria.

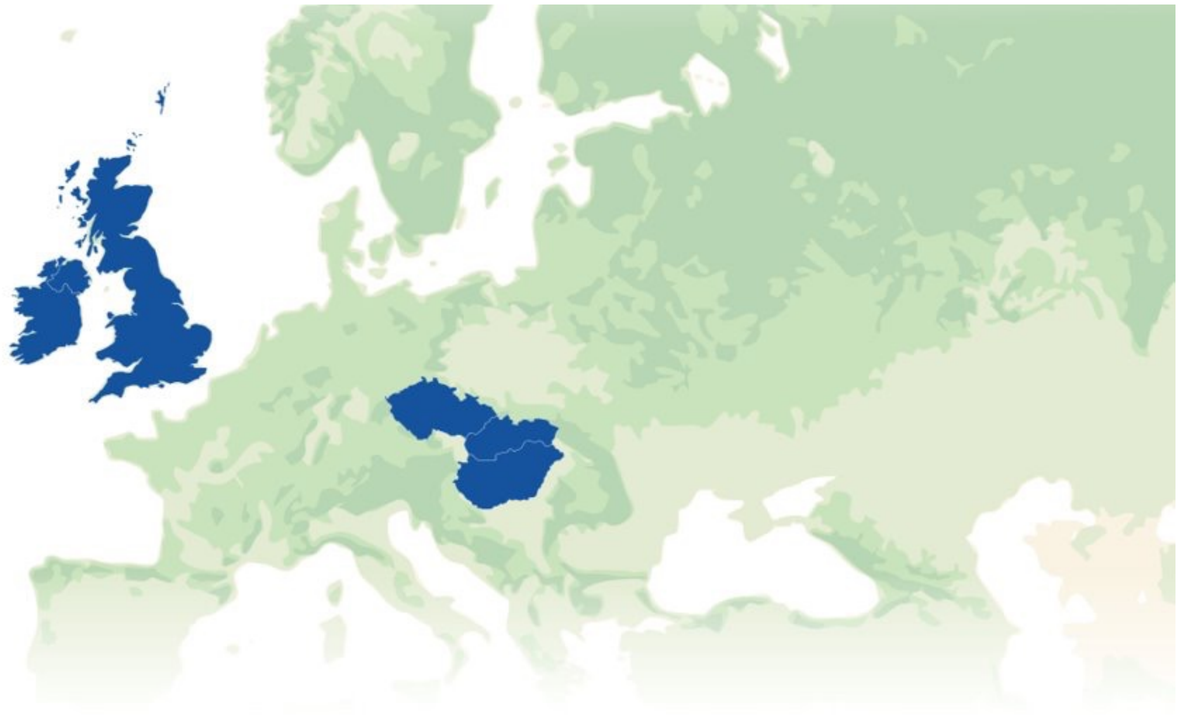
An introduction to Reinforcement Learning for pricing.

Cesc Cunillera

TESCO

What is Tesco

Our businesses



Key facts

£57.7bn
Group Sales[1]
2022: £54.8bn

£2,630m
Group adjusted operating profit[2]
2022: £2,825m

£2,133m
Retail free cash flow[3]
2022: £2,277m

Data Science at Tesco

Team

Growth in the team size over 6 years approaching a total team size of 120 Data Scientists and ML Engineers

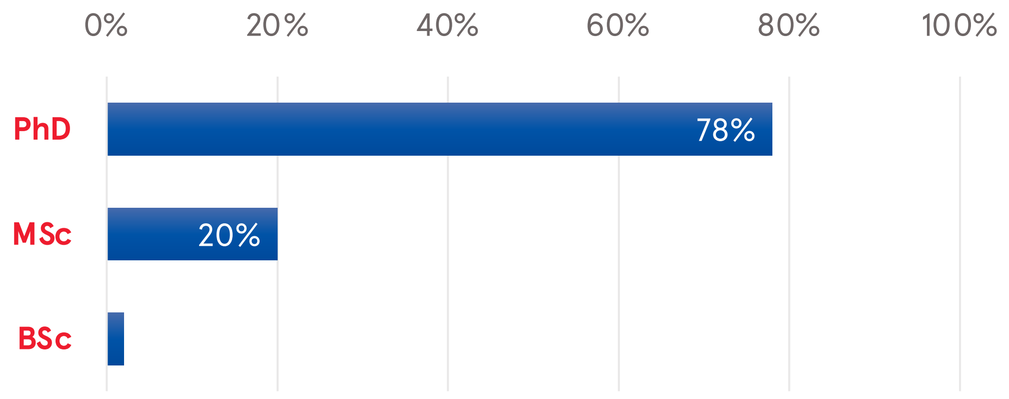
Locations

Expanded from 1 to 4 global locations
Welwyn, London, Krakow, Bengaluru

Products

Supporting 30+ Technology Products between algorithmic features and data science platforms

Academic Background

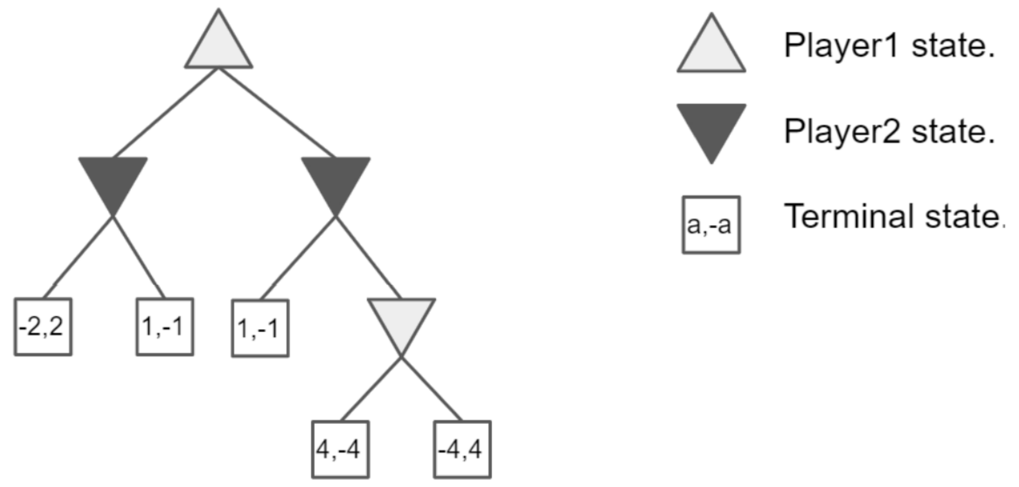


Collaborating with
Top Universities

Overseeing
PhDs
MSc Projects
PhD Internships

OR President's medal

An introduction to RL for pricing

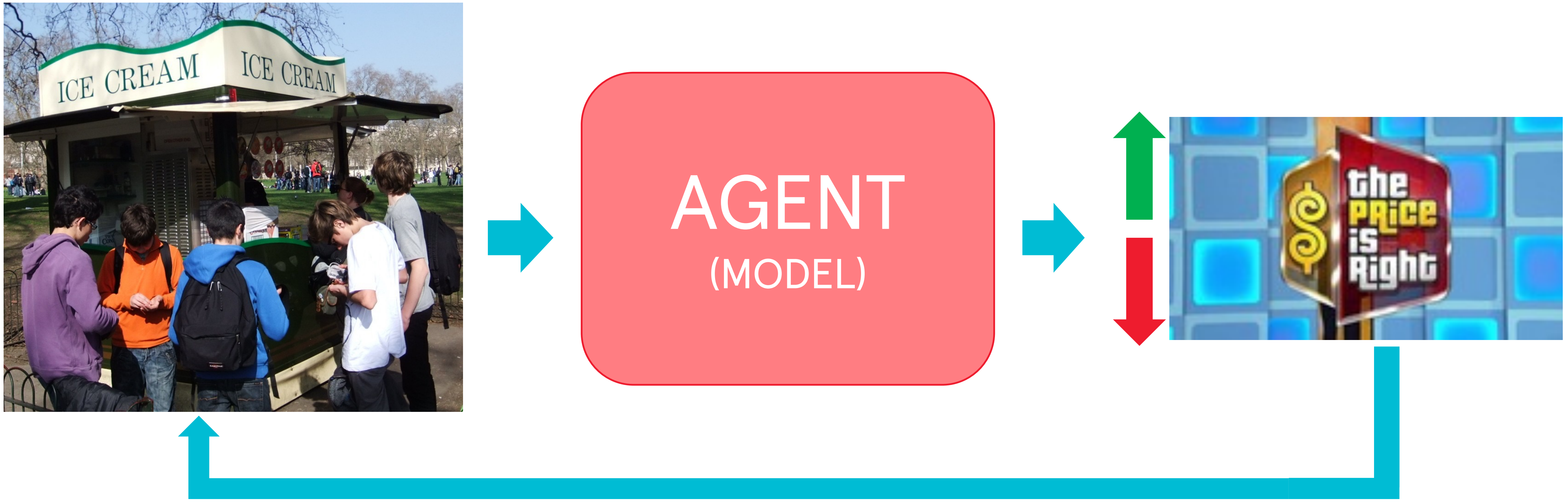


Taken from arXiv:2111.05884

The basics.

States, MDPs and policies.

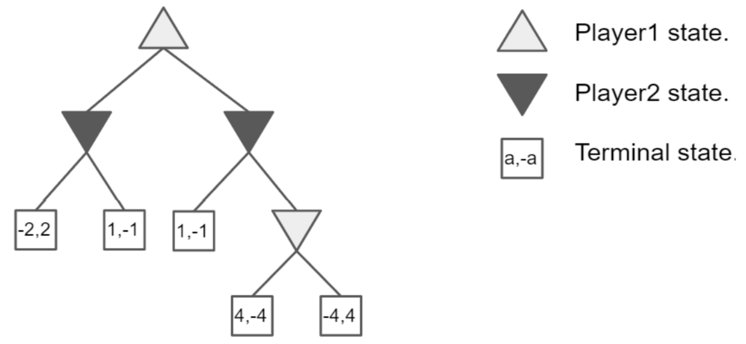
States and observations



The idea behind reinforcement learning is to gather <u>observations</u> from the environment and train an <u>agent</u> to make appropriate decisions based on those observations.

We assume the world is a chain of Markovian <u>states</u>. We can gather observations and interact with the world through our decision making process acting on these states. The outcome of this interactions is the agent’s <u>policy</u>.

Markov Decision Processes (MDP)



Taken from arXiv:2111.05884

To define an MDP we need $(S, A, T, r, \gamma, \mu)$:

- A state space (S) or an observation space $(\mathcal{O})$, which could be finite or infinite.
- An action space (A), which could be discrete or continuous (infinite).
- A transition function (model) (T) where $T\colon\ S \times A \to \Delta(S)$, with $\Delta(S)$ the space of probability distributions of S.
- A reward function (r) where $r\colon\ S \times A \to [0,1]$ (WLOG).
- A discount factor γ with $0 \leq \gamma \leq 1$.
- An initial state distribution $\mu \in \Delta(S)$, which we will often take to be supported only on some initial state $s_0 \in S$.

A trajectory τ is composed of state-action-reward triplets: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$.

A policy π is a mapping from the space of all trajectories (of any length) $(\mathcal{H})$ to the space of distributions of A,

$$\pi\colon\ \mathcal{H} \to \Delta(A)$$

Given these we can define a critical object in RL, the Q-value function

$$Q^{\pi}(s,a) \coloneqq \mathbb{E}_{\tau\sim\pi}\left[\sum_t \gamma^t\, r(s_t, a_t) | s_0 = s, a_o = a\right]$$

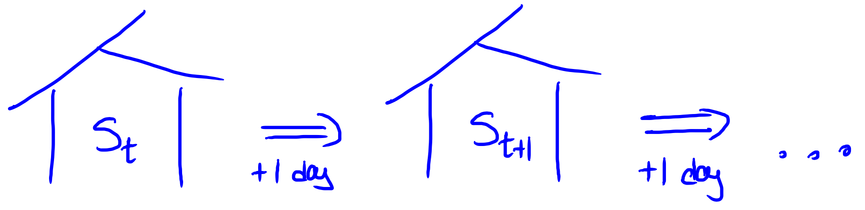
Note that $0 \leq Q^{\pi} \leq 1/(1-\gamma)$.

Gelateria MDPs

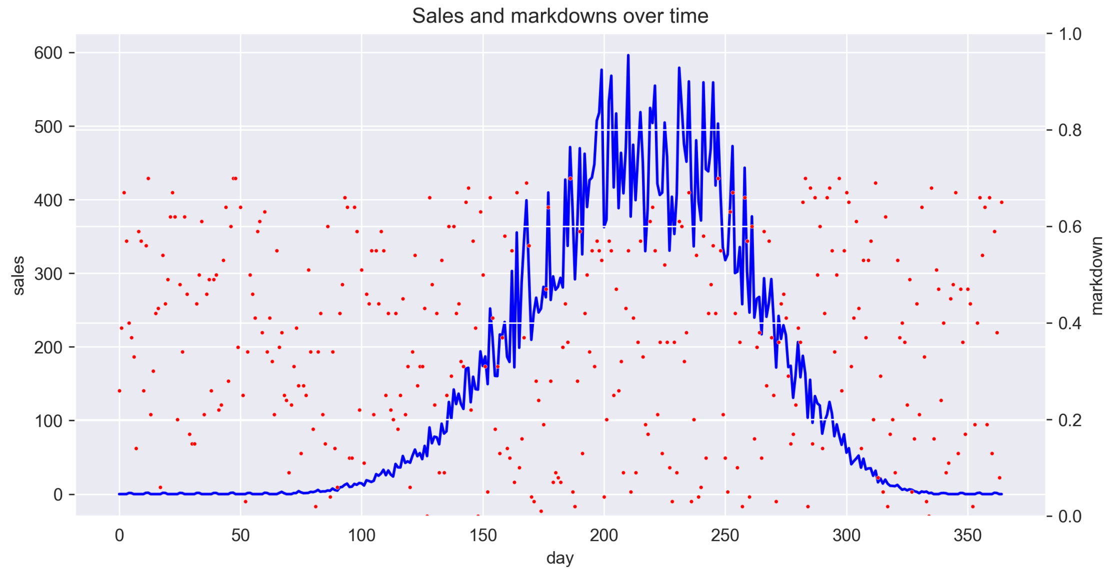
Ben has been working on his gelato for the past 10 years, and he believes that there should be a more efficient way of pricing his products. He is also thinking of expanding his gelato emporium. How to price Ben's gelato?

For the purposes of coming up with a solution, we have been told that:

- Ben sells N different gelato flavours, and makes more gelato every Sunday after closing the gelateria.
- To avoid confusing customers, Ben wants to fix the prices of the gelatos before opening every morning.
- Ben is willing to allow reductions (markdowns) in the range $[0, 1)$.
- Ben loves data, and he has been collecting daily sales data since he opened his store.



- observations : sales, stock, date, weather, ...



Policies

The end-goal of the RL pricing problem is to come up with a policy that maximises returns subject to some constraints (*e.g.* we might want to penalise excessive waste, taking too many actions, …), which we encode in a reward function r,

$$\pi_*(s) \coloneqq argmax_{a\in A}\, Q(s,a) \coloneqq argmax_{a\in A}\, \mathbb{E}\left[\sum_t \gamma^t\, r(s_t, a_t) | s_0 = s, a_o = a\right]$$

Caveats:

- In some cases, not all constraints can be encoded into the reward function. For example, there might be a posteriori business/legal logic constraints that cannot be easily put in the form of a Lagrange multiplier.
- Constrained optimisation is exactly solvable in convex problems. For non-convex problems all hell can break loose, and most real world applications are non-convex.
- The problem we set out to solve depends on our knowledge of the transition matrix T, this can be a big bottleneck.

The gelato pricing problem

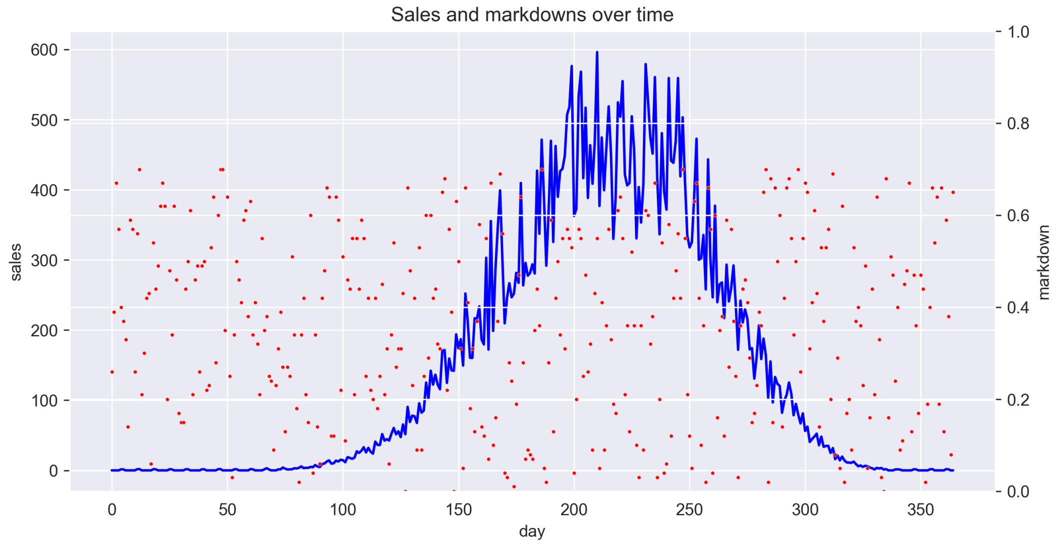
Data concerns:

- Accuracy of the historical data.
- Amount of parameter space explored, e.g. it's hard to convince a business to sell products at 1% of its value to "gather data".
- Access to the underlying generative process.

Computational concerns:

There are two quantities that define the dimensionality of the problem: $(|S|, |A|)$. For us, the action space for a product is fixed, $|A| = 101$, and the dimensionality of the problem is $\mathcal{O}(100 \times |S|)$. The size of the state space could be the number of different gelato flavours in the store. However, the dimensionality of S can grow very fast.

We will see that the approach chosen to solve the problem can be mapped back to $|S|$.



RL and pricing.

What makes (retailer) pricing different?

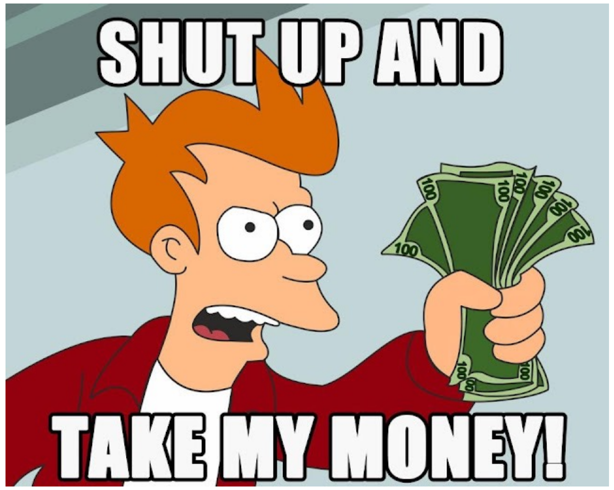
Partially Observable MDP

We don’t have access to the sales generative process.

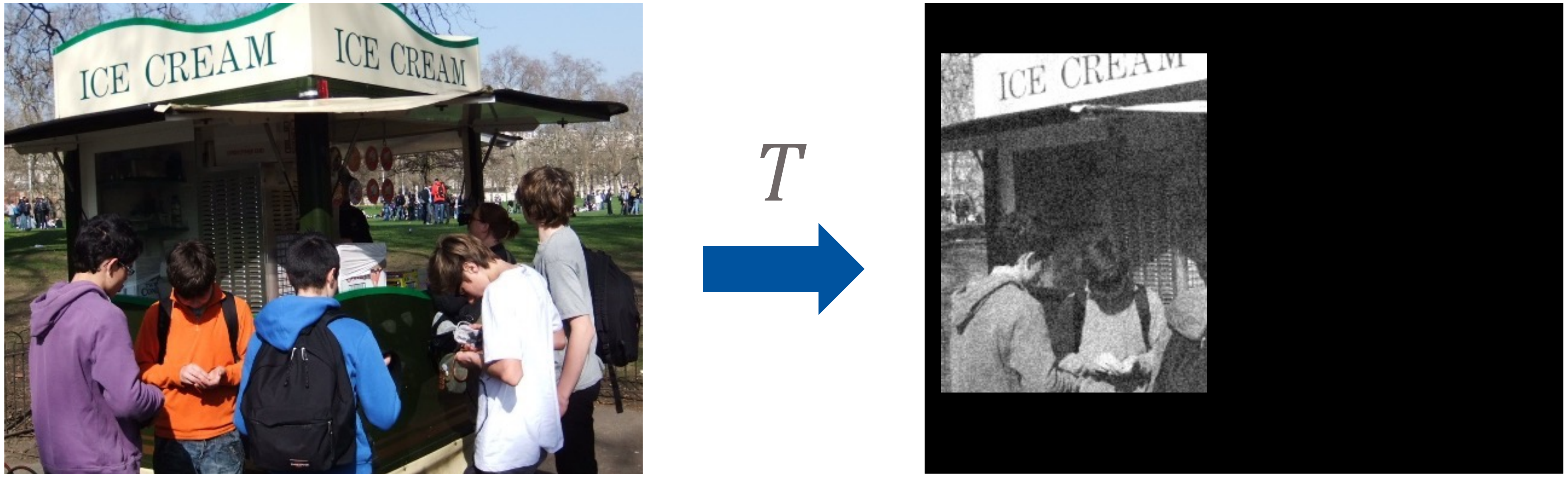
In a videogame, *from any snapshot* we can tell what will happen in the next frame.

When pricing products, this is not the case.

Taken from arXiv:1312.5602v1



The gelato pricing problem II



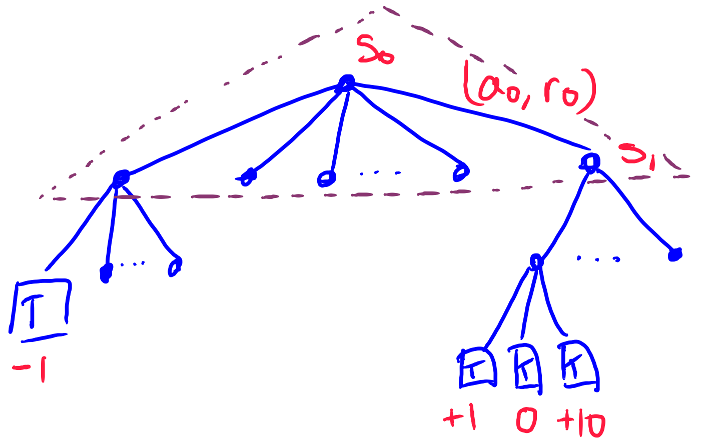
We will (mostly) sweep this under the rug and concentrate on given some T finding a solution to the pricing problem.

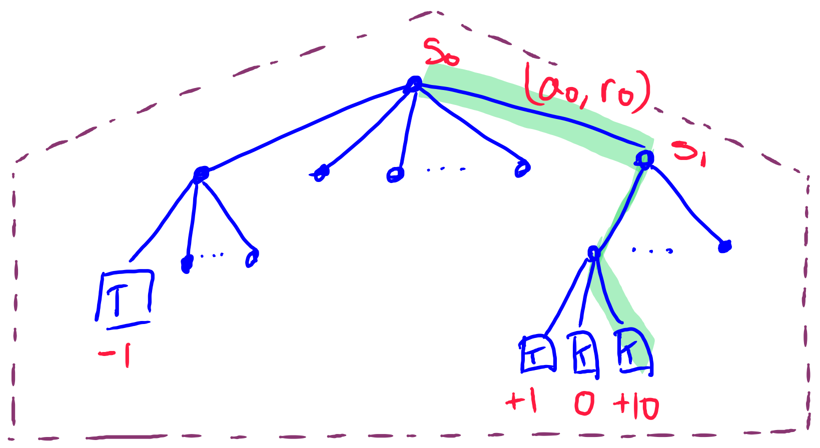
In principle, there are ways to ameliorate the issues with T usually through exploration (e.g. trialling new reduction ranges) or limiting the action of the agent to the support of T.

Tabular methods.

Tabular problems

We consider problems with $|S|$ small enough so that we can walk the entire tree and find the global optimum.





In this case, the ideal solution is a dynamic programming (DP) approach where we compute the Q-value function for every branch in the tree and choose the maximal one.

The key limitation is time complexity and space complexity. Tree walks are slow (even if you are smart about DP) as you must effectively visit every node once.

The Q-value function has $\mathcal{O}(100\times|S|)$ entries. Doing some sort of beam search over reductions and $|S|$ to keep the (p, q) top choices reduces the dimensionality to $\mathcal{O}(p\times q)$. If we price n-steps ahead this grows like $(p\times q)^n$, not ideal.

Two directions

The underlying assumption of (most) DP approaches is that we have access to the true underlying generative process such that the reward is always perfectly known. In most situations, the reward is better described as a random variable. Further, the number of states is often too large to describe in a discrete way (w/o starting to make use of approximations).

Push in two main directions:

1. Time – More efficient exploration: learn from MC approaches.
2. Space – "Better representations" for the Q-value function: use of Q-value function/policy approximators (e.g. neural networks).

K-armed bandit problems

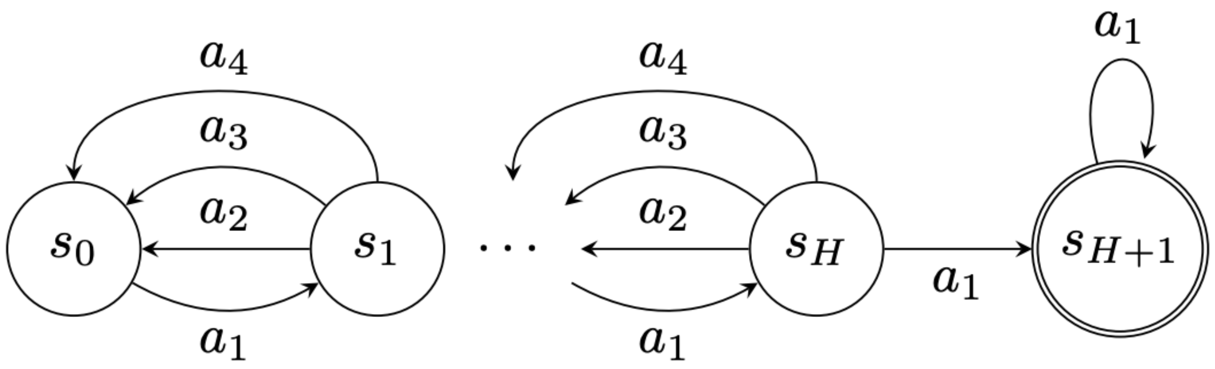
An MDP with $\gamma = 0$ reduces to a K-arm bandit problem, where maximising the reward over H steps is equivalent to minimising the regret

$$R \coloneqq H \cdot argmax_i \, \mu_i - \sum_{h=0}^{H-1} \mu_{a_h}$$

with μ_i the mean reward for taking action $i \in A$, or equivalently pulling the *i-th* arm. Henceforth, we consider the reward to be a random variable.

In this setting, the probability of minimising the regret with random exploration starting from an arbitrary state can be exponentially small, $\mathcal{O}(|A|^{-H})$. For our gelateria, we could expect anywhere $\mathcal{O}(10^7) - \mathcal{O}(100^7)$ configurations per product.

We <u>need</u> strategic exploration.



Taken from: Agarwal, A., et al. “Reinforcement Learning: Theory and Algorithms.”

Upper Confidence Bound (UCB)

† Note one can show that with probability $(1-\delta)$ bounds the regret in H steps to (roughly)

$$R_H \sim \mathcal{O}\left(\sqrt{H|A|\log\left(\frac{H|A|}{\delta}\right)} + |A|\right)$$

Similar bounds exists for LinUCB [Dani et al., 2008]

For a K-armed bandit, we can efficiently explore† the state space by making use of UCB (or LinUCB for continuous spaces).

The idea is to keep track of the mean reward and promote taking actions that have been not explored much. In practice, we try every action once and then choose for $t = |A| + 1, \dots, H$

$$a_t = argmax_{a \in A}\left(\hat{\mu}_a + \sqrt{\frac{\log\left(\frac{H|A|}{\delta}\right)}{N_{a,t}}}\right)$$

where $\delta \in (0,1)$ and

$$\hat{\mu}_a = \frac{1}{N_{a,t}}\left(r_a + \sum_{i=|A|}^{t-1} \mathbf{1}\{a_i = a\}\, r_i\right) \qquad N_{a,t} = 1 + \sum_{i=|A|}^{t-1} \mathbf{1}\{a_i = a\}$$

Beyond tables.

MC-type exploration and policy approximation.

MC Control

With UCB, we are not making any assumptions regarding the generative process or our access to the state, but we still have to explore every action available at least once.

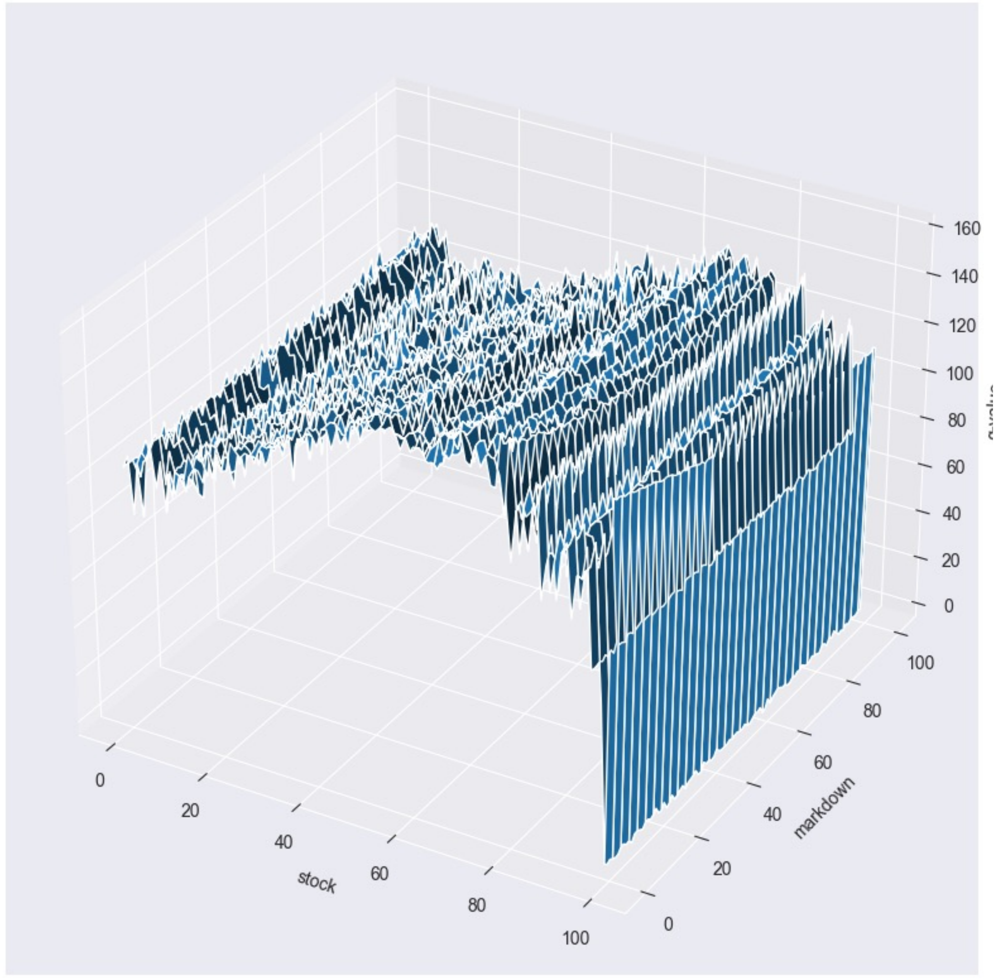
The goal of MC control is to update the value function directly from sampled trajectory as

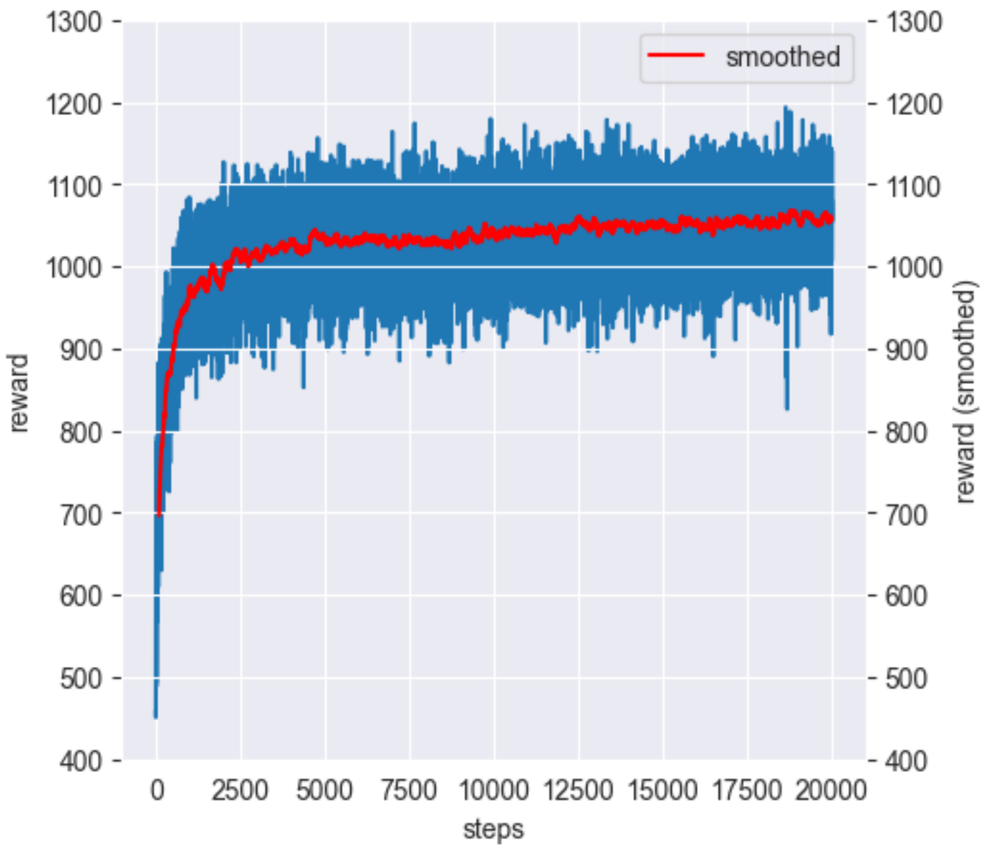
$$Q^{\pi} \leftarrow \mathbb{E}_{\pi}\left[\sum_{t} \gamma^{t}\, r(s_t, a_t)\right]$$

For a basic MC control algorithm:

1. Choose any arbitrary ϵ-soft policy π_ϵ to sample the state space
2. For N episodes w/ horizon H, generate a trajectory $\tau_{\pi_\epsilon} \sim s_0, a_0, r_0, \dots, s_{H-1}, a_{H-1}, r_{H-1}$
3. For each step $h = H-1, \dots, 0$ compute:
   1. Update visits to state-action pair: $N(s_h, a_h) \leftarrow N(s_h, a_h) + 1$
   2. Compute the discounted rewards: $G(s_h, a_h) \leftarrow r_h + \gamma G(s_h, a_h)$
   3. Update the Q-value function and the target policy: $Q^{\pi}(s_h, a_h) \leftarrow Q^{\pi}(s_h, a_h) + \frac{1}{N(s_h, a_h)}[G(s_h, a_h) - Q^{\pi}(s_h, a_h)]$ $\pi(s_h) \leftarrow argmax_{a \in A}\, Q^{\pi}(s_h, a)$

Gelateria MC Control





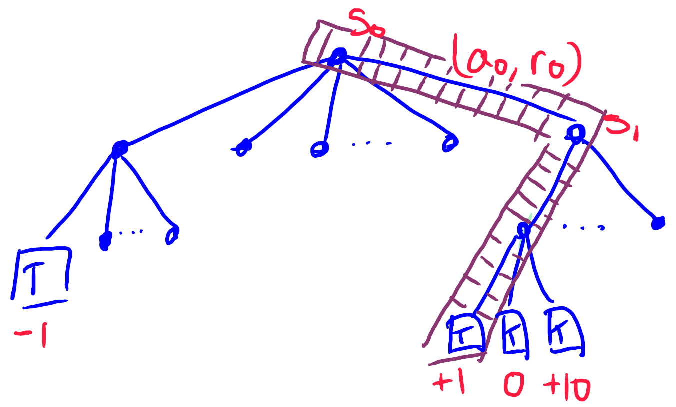
Temporal-Difference (TD) methods

MC control requires sampling the entire trajectory. An alternative to MC control for infinite horizon processes is given by TD methods.

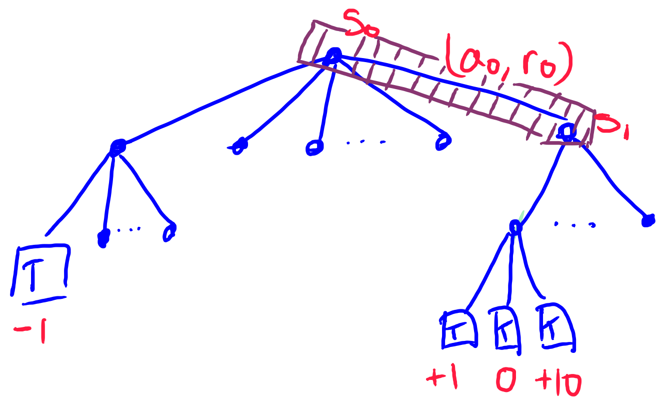
Now the goal is to update value function by making use of the optimality equation

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma\, Q^{\pi}(s_{t+1}, a_{t+1})$$

MC control and TD represent a (bias, variance) trade-off. MC= (↓,↑), TD= (↑,↓). In general, these two methods approach the same solution, although TD is more sensitive to initialisations.

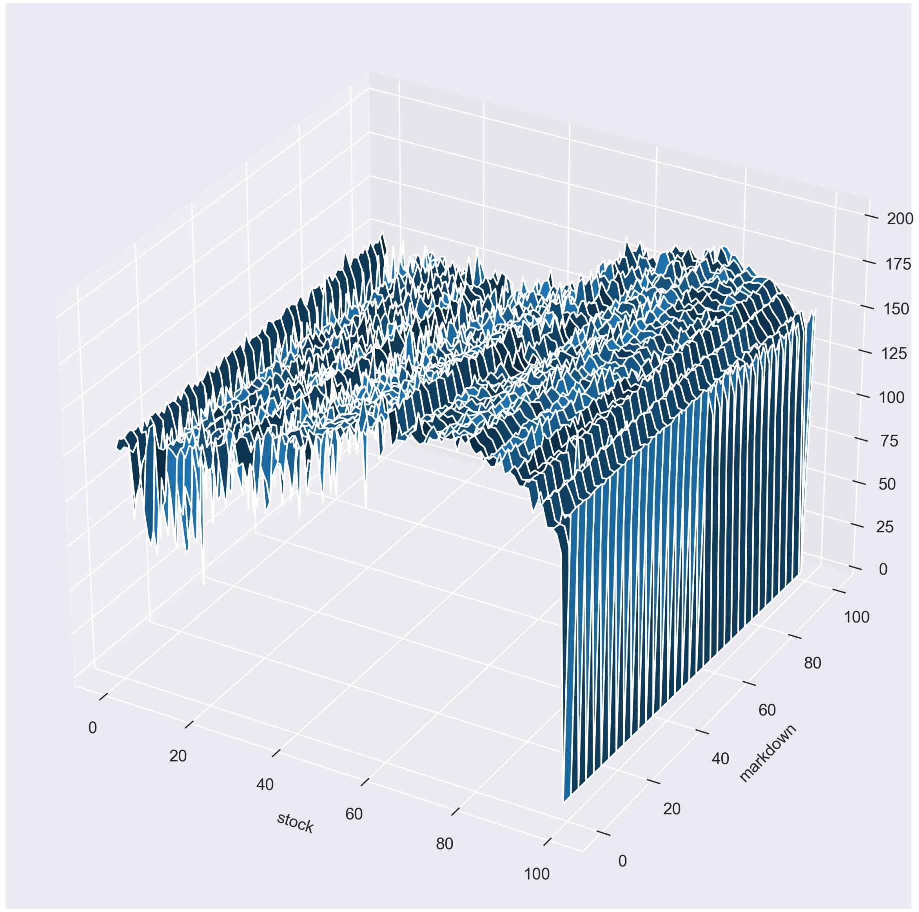


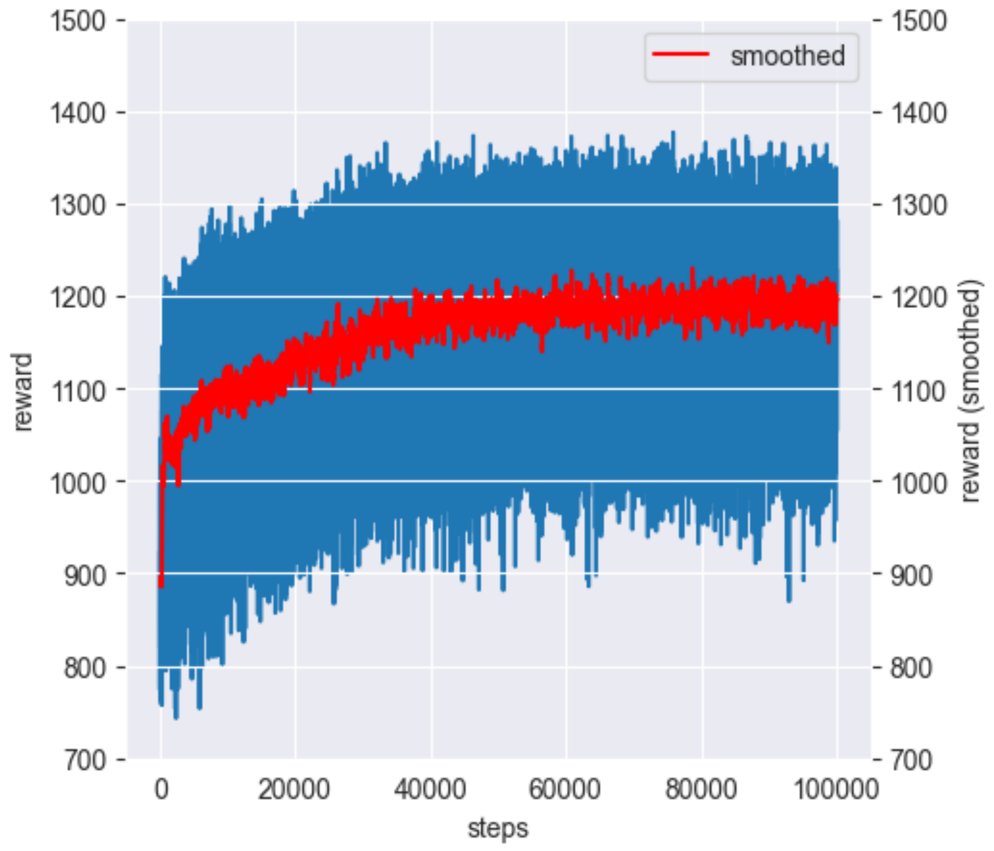
MC Control roll-out



TD(0) method

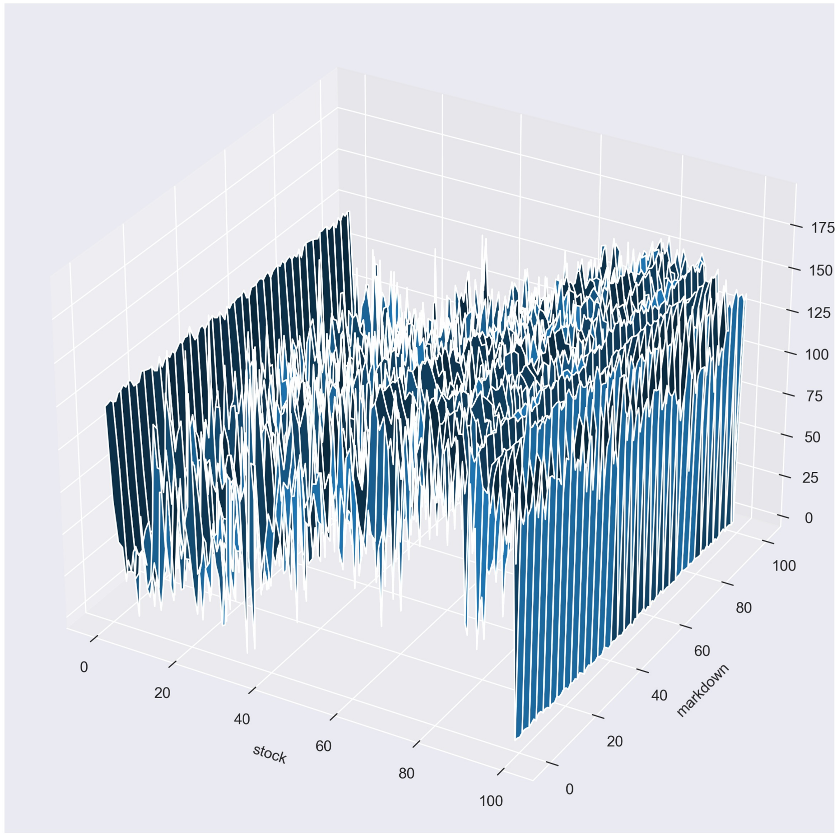
Gelateria TD(0)

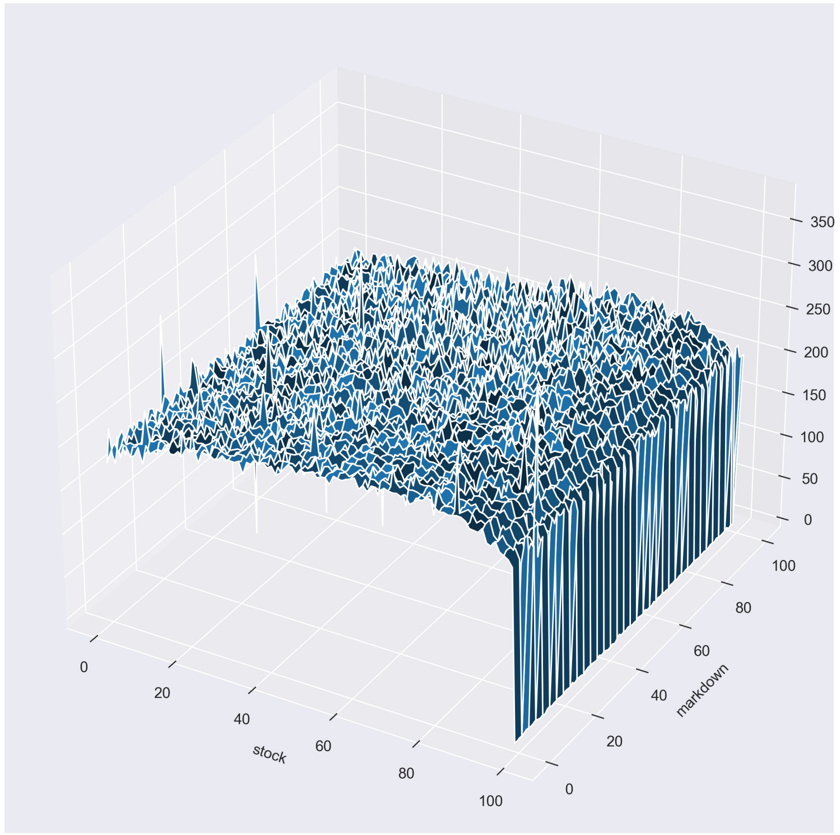




TD(0) initialisation

Q-values after 10.000 steps





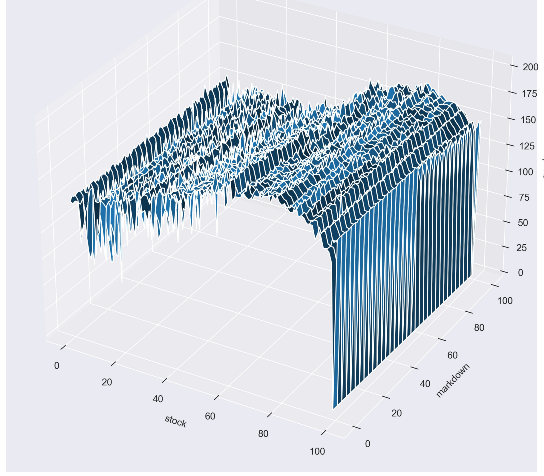
“Bad” init.

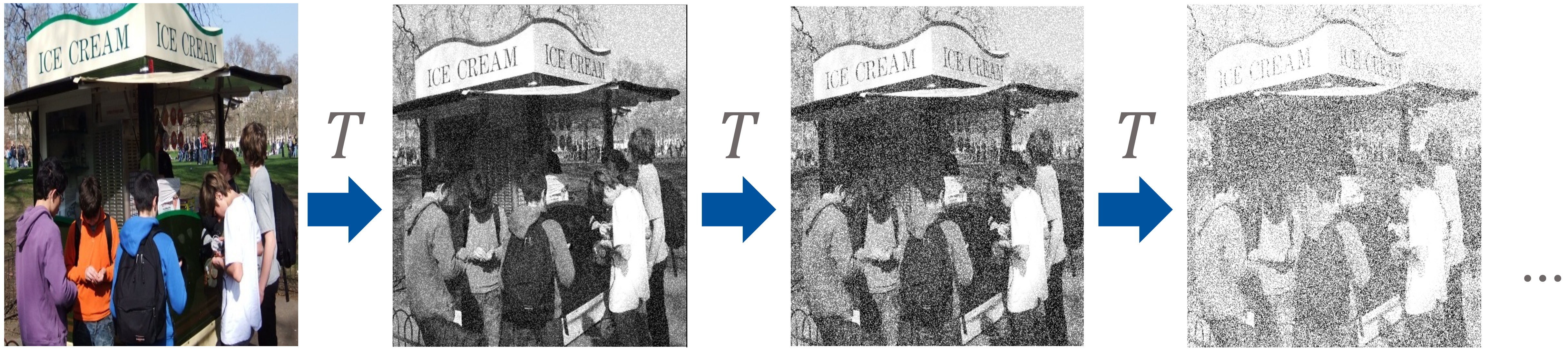
$$Q_0 = f(x, y) \mid f(0, y) = 10, f(100, y) = 0$$
$$\forall\, y \in [0,100]$$

Random init.

$$Q_0 = \mathcal{N}(0,1)$$

(Not-so-real) World dynamics





By repeatedly applying some approximation T to the real world transitions $\hat{T}$ are losing information about the true distributions. Best to avoid, for example storing historical trajectories in a replay buffer $\mathfrak{B}$.

DQN

Taken from arXiv:1312.5602v1

DQN is an example of a TD method combining a replay buffer with a function approximation for the policy. It *should* have the same convergence/optimality guarantees as other TD methods but with the advantage of admitting *arbitrarily large* state-action spaces.

We generate the target

$$T^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma\, Q^{\pi}(s_{t+1}, a_{t+1})$$

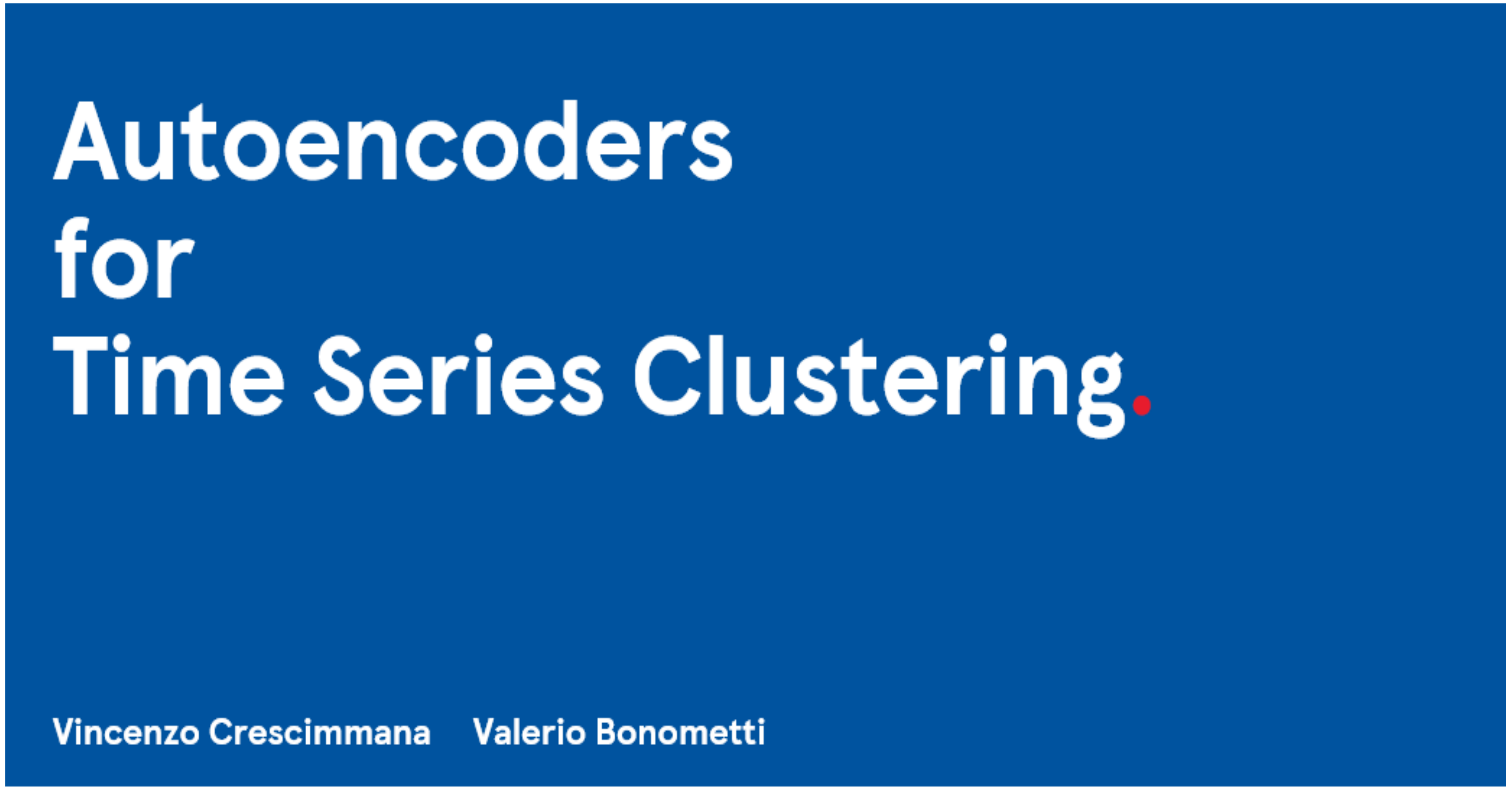
where Q^{π} is the output of some deep network, and the nets are trained to minimise the error in

$$\|(T^{\pi} - Q)(s,a)\|_2$$

Further content

To learn how to use AEs for learning time series embeddings see

and keep an eye open as we are working on extending and (hopefully) open sourcing this project providing a toy environment for pricing, with pseudo data generation and some prediction and optimisation models.



Autoencoders for Time Series Clustering.

Vincenzo Crescimmana Valerio Bonometti

Sat, 3rd of June (soon to be on YouTube)

(Some) Useful resources

RL books:

- Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). The MIT Press.
- Agarwal, A., Jiang, N., Kakade, S. M., & Sun, W. (2021). Reinforcement Learning: Theory and Algorithms.

Deep learning and reinforcement learning courses (w/ video lectures available on YouTube):

- CMU Deep Learning 11785 – Link to course
- Berkeley Deep Reinforcement Learning CS285 – Link to course

Websites:

- RLLib Algo docs (compendium of common algos w/ papers and implementations).
- PyTorch Lightning Bolts for RL (implementations of Deep RL models in Lightning).
- Stable Baselines (implementations of Deep RL w/ many gym environments).

Thank you.

TESCO