intermediate periods. As the time horizon is fixed, and saving implies a gross rate of return of exactly 1, the dimension of the state space satisfies $|\mathscr{S}| = T \cdot (\bar{s} + 1)$, while the action space satisfies $|\mathscr{A}| = M + 1$. Effectively, the set of *feasible* actions conditional on the state, s_t, satisfies $|\mathscr{A}(s_t)| = s_t + 1 \leq \mathscr{A}$.

With the purpose of learning the true value function $V(s_t)$ for a given state s_t, we let our agent be an ϵ-greedy Q-learner. In order to illustrate how the Q-learner learns the value function, it is instructive to define the *true value map* for the problem, $\mathcal{Q}(s_t, a_t) \mapsto \mathbb{R}$, that maps a given state and action into a value as also discussed in the previous section. Hence, we may simply think of the relation between the value function and the value map as $V(s_t) = \max_a \mathcal{Q}(s_t, a_t)$. That is, the value function implicitly selects the action $a^*(s_t)$ in the set of feasible actions $\mathscr{A}(s_t)$ that maximizes $\mathcal{Q}(s_t, a_t)$ holding s_t fixed.

In order to learn $\mathcal{Q}(s_t, a_t)$, the agent is equipped with a matrix Q of size $|\mathscr{S}| \cdot |\mathscr{A}|$ with all elements initialized to some values - in this case, we simply choose 0. Following an updating rule, the agent explores the environment and gradually updates the matrix Q in order to approximate $\mathcal{Q}(s_t, a_t)$. As Q tends to $\mathcal{Q}(s_t, a_t)$, the learner implicitly approximates the optimal actions for each state, s_t.

In the next subsection, we provide a a simple example of a learning algorithm with an ϵ-greedy Q-learner for the matrix Q that works well in discrete and bounded cases. As we move forward, we will introduce other methods for learning such representations, all of which either intend to approximate $\mathcal{Q}$, intend to approximate the optimal action $\mu^*(s_t)$, or does both at the same time.

2.2 Updating the Q-matrix

With probability ϵ, the agent chooses a random feasible action in the time-t available action space $\mathscr{A}(s_t) \in \mathscr{A}$. This kind of choices serve to *explore* the action space. Alternatively the agent chooses the action such that $a_t = \arg\max_{a_t} Q(s_t, a_t)$, in which case the agent *exploits* it's current knowledge to improve certainty on the optimal path. Learning will be the process by which the agent updates the values in Q to make better desicions in the $1 - \epsilon$ cases where it chooses the optimal choice. Updating Q is an iterative process, and essentially follows the idea of a Bellman equation, augmented by a learning rate $\eta \in (0, 1)$. That is, we update Q according to:

$$Q(s_t, a_t) \leftarrow (1 - \eta)Q(s_t, a_t) + \eta(r_t + \beta \max_a Q(s_{t+1}, a)) \tag{6}$$

where r_t is the instantaneous reward $u(a_t)$ and β is the discount factor [19]. To fully train the model we repeatedly expose the Q learner to the environment over k *episodes*, letting it choose actions and reap rewards while updating the Q-matrix.

Figure 1 plots the learned value map when solving the model given in (4). In period 0, no information is learned for $s_0 < 100$ as this initial condition is fixed - thus, we only show values for $s_0 = 100$. The highest value is achieved from choosing $a_0 \approx 40$. The need to include the time step as a state becomes apparent when looking at panel C. Because the continuation value is 0 for all choices in this period, the reward is independent of the remaining stock.

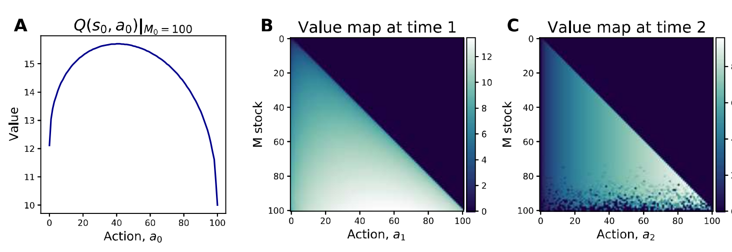

Fig 1. Per-time period valuemaps learned by the Q-learner in 50.000 episodes with $\bar{s} = 100$, $T = 3$, $\eta = 0.6$ and $\beta = 0.9$. We take $\epsilon = 0.5$ to excessively explore the state-action space. Note in panel A we only show values for $s_0 = 100$.

In figure 2, we show the number of times the algorithm visits each field in Q during the 50,000 episodes that we used to generate figure 1, highlighting the usefulness of relying on nonconverged versions of Q for guiding the exploration of the state-action space.

Unlike in classical dynamic programming algorithms, the Q-learner requires fixing hyperparameters to achieve good convergence to the true value map. In particular, the number of learning episodes, the