Validating Logistic Q-Learning

Sebastian Markgraf
Student Computer Science
Karlsruhe Institute for Technology
Karlsruhe, Germany
sebastian.markgraf@student.kit.edu

Philipp Becker*
ALR Institute
Karlsruhe Institute for Technology
Karlsruhe, Germany
philipp.becker@kit.edu

Onur Celik*
ALR Institute
Karlsruhe Institute for Technology
Karlsruhe, Germany
onur.celik@kit.edu

Maximilian Hüttenrauch*
ALR Institute
Karlsruhe Institute for Technology
Karlsruhe, Germany
maximilian.huettenrauch@kit.edu

* Equal contribution, ordered alphabetically

***Abstract*—Logistic Q-Learning was presented in 2020 as an extension to the Relative Entropy Policy Search. The main extension consists of the introduction of the Q function and therefore motivates the name QREPS. Additionally, the original work adds the empirical logistic bellman error as alternative to the squared bellman error.**

As the original implementation was not public at the beginning of this work and the results seem to be promising, this work implements the algorithm and verifies the results of the original work. The focus lies on the implementation of REPS and QREPS and the performance on NChain and Cartpole.

All related code is provided on GitHub by the authors.

***Index Terms*—REPS, QREPS, Reinforcement Learning**

I. Introduction

Reinforcement Learning is currently a hot research topic. A lot of the approaches focus on the mastering of games, due to their strictly defined goals and rules. Especially, the achievements of DeepMind (DM) and OpenAI of beating human professional players in their respective games made reinforcement learning popular [1, 2]. But the application of reinforcement learning is way more diverse and includes for example robotics [3] and financial trading [4].

Reinforcement learning in robotics often uses policy search algorithms due to the ability to handle continuous action spaces [3]. These algorithms introduce constraints to the possible policies and then perform numerical optimization for the optimal policy. In many other cases this approach is unfeasible and is therefore less known than the easy to use Q-Learning and its deep variants [5, 6].

Relative Entropy Policy Search (REPS) [7] was presented with guarantees for convergence to the optimal policy, but is limited by the dependency on the transition model, which in addition introduces a biased estimate of the policy evaluation step. Logistic Q-Learning (QREPS) [8] extends the original REPS idea and resolves the dependency on the transition model. This work summarizes the findings from implementing both REPS and the newly presented algorithm from [8].

II. Fundamentals

QREPS relies on a range of reinforcement learning fundamentals, which are presented in the following.

A. Markov Decision Process

A Markov Decision Process (MDP) as introduced in [9] is defined as a tuple

$$(\mathcal{X}, \mathcal{A}, r, \mathcal{P}, \nu_0).$$

An MDP formalizes the interaction between an agent and an environment. The environment is always in a state $x \in \mathcal{X}$, where ν_0 describes the initial distribution of the states. The agent chooses an action $a \in \mathcal{A}$ and executes it. The environment then transitions according to $\mathcal{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to [0, 1]$ into the next state $x' \in \mathcal{X}$. At the same time the agent receives a reward according to the reward function $r : \mathcal{X} \times \mathcal{A} \mapsto \mathbb{R}$ [9].

Often, the agent does not observe the state of the environment directly but features created from a feature function ψ. Additionally, we define a state-action feature function denoted by φ.

Depending on the environment, an MDP can be defined as finite or infinite. If the MDP is finite the environment only runs until a set amount of timesteps or until a stop condition is met. Especially for control problems, the MDP can be defined as being infinite and not reaching a termination. We only discuss finite problems in this work. To transform an infinite problem into a finite problem, usually one considers episodic runs of the environment.

When saving the states, actions and following states one can construct a history of interactions as a trajectory of transitions $\xi_i = (X_i, A_i, X_i')$ [8]. To account for the distance in time of rewards, we need to introduce a discount factor $\gamma \in (0, 1)$ that discounts rewards that are further away. Otherwise rewards that are extremely far in the future are weighted the same as rewards that are gathered in the first step. In that case the agent could potentially aim for the future reward without ever

reaching it. With the help of the discount factor the normalized discounted return is defined

$$R = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r(x_t, a_t)\right].$$

The goal of reinforcement learning is now to learn a policy $\pi : \mathcal{X} \times \mathcal{A} \to [0, 1]$ that optimizes the normalized discounted return as defined above. A policy is a mapping of states to a probability distribution over the possible actions that allows to describe the decisions and behaviour of the agent. To get a decision from the policy, an action a is sampled from the policy as $a \sim \pi(\cdot|x)$ for the state x.

An equivalent definition is the "normalized discounted state-action occupancy measure" [8]

$$p(x, a) = (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}_{(x_t,a_t)=(x,a)}\right]$$

which maximizes the discounted return. This occupancy measure implicitly defines a policy

$$\pi_p(a|x) = \frac{p(x, a)}{\sum_{a'} p(x, a')},$$

which yields $p(x, a)$ when run. To shorten the notation with these occupancy measurements, we introduce the following operators with the notation of [8]:

$$\begin{aligned}(P^T p)(x') &= \sum_{x,a}\mathcal{P}(x'|x, a)p(x, a)\\ (E^T p)(x) &= \sum_{a} p(x, a)\\ (\Psi^T(E^T p)) &= \sum_{x}\psi(x)(E^T p)(x)\\ (\Phi^T p) &= \sum_{x,a}\varphi(x, a)p(x, a)\end{aligned}$$

Here P^T specifies the occupancy measure of the following states for the state-action occupancy measure p. E^T simplifies the notation of the state occupancy measure instead of summing over all actions, since $p \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ is defined over the states and actions. Ψ^T and Φ^T allow to easily describe the distribution of the features according to the corresponding feature function ψ and φ.

The problem of finding an optimal occupancy measure is then formulated as the following linear program (LP)

$$\begin{aligned}\underset{p\in\mathbb{R}^{\mathcal{X}\times\mathcal{A}}}{\text{maximize}} \quad & \langle p, r\rangle\\ \text{s.t.} \quad & E^T p = \gamma P^T p + (1-\gamma)\nu_0.\end{aligned} \tag{1}$$

B. Relative Entropy Policy Search

Peters et al. present REPS in [7]. The main idea of REPS lies in bounding the loss of information between the state-action distributions measured by the relative entropy. This is measured by calculating the Kullback-Leibler Divergence (KL)

$$\begin{aligned}D(p||p_k) = \sum_{x,a}\Bigg(&p(x, a)\log\frac{p(x, a)}{p_k(x, a)}\\ &- p(x, a) + p_k(x, a)\Bigg)\end{aligned} \tag{2}$$

between the observed data distribution $p_k(x, a)$ and the data distribution $p(x, a)$ generated by the new policy π and denoting the upper bound by ε.

By introducing the constraint on the relative entropy to the LP in Eq. (1) and relaxing the equivalence to only require feature equivalence we gain the LP for REPS as

$$\begin{aligned}p_{k+1} = \underset{p\in\mathbb{R}^{\mathcal{X}\times\mathcal{A}}}{\text{maximize}} \quad & \langle p, r\rangle\\ \text{s.t.} \quad & \varepsilon \geq D(p||p_k)\\ & \Psi^T E^T p = \Psi^T(\gamma P^T p + (1-\gamma)\nu_0).\end{aligned}$$

These equations can be solved by minimizing the corresponding dual function

$$g(\theta) = \eta\log\left(\sum_{x,a} p_k(x, a)\exp\left(\varepsilon + \frac{\delta_\theta(x, a)}{\eta}\right)\right)$$

and setting our policy as

$$\pi_{k+1}(a|x) \propto \pi_k(a|x)\exp\left(\frac{\delta_{\theta^*}(x, a)}{\eta^*}\right).$$

Neu et al. [10] add a variation on REPS where the relative entropy is not constrained but gets regularized. In that case REPS can be seen as Mirror Descent algorithm with the following formulation [8]

$$\begin{aligned}p_{k+1} = \underset{p\in\mathbb{R}^{\mathcal{X}\times\mathcal{A}}}{\text{maximize}} \quad & \langle p, r\rangle - \frac{1}{\eta}D(p||p_k)\\ \text{s.t.} \quad & \Psi^T E^T p = \Psi^T(\gamma P^T p + (1-\gamma)\nu_0).\end{aligned}$$

To optimise this formulation ε is removed from the dual and the regularization is done by fixing η.

Despite the elegant formulation, REPS was not used much due to the dependency on having the transition model, when running in stochastic environments. In most complex environments the transition model is not available and would need to be estimated.

III. Logistic Q-Learning

Bas-Serrano et al. introduce their algorithm as Logistic Q-Learning (QREPS) in [8]. QREPS has the same underlying idea as REPS of decomposing the LP in Eq. (1) and bounding the relative entropy between following state-action distributions with the KL (Eq. (2)). But instead of the introduction of only one distribution p as seen in REPS, the authors of QREPS add a "mirror image of p" [8] named $d \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$

$$\begin{aligned}\underset{p,d}{\text{maximize}} \quad & \langle p, r\rangle\\ \text{s.t.} \quad & E^T d = \gamma P^T p + (1-\gamma)\nu_0\\ & d = p \qquad p \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}, d \in \mathbb{R}_+^{\mathcal{X}\times\mathcal{A}}.\end{aligned}$$

To optimise this LP we use the dual formulation of the problem

$$\begin{aligned} \underset{V\in\mathbb{R}^{\mathcal{X}},Q\in\mathbb{R}^{\mathcal{X}\times\mathcal{A}}}{\text{minimize}} \quad & (1-\gamma)\langle\nu_0, V\rangle \\ \text{s.t.} \quad & Q = r + \gamma PV, \qquad EV \geq Q. \end{aligned}$$

As [8] points out, this formulation naturally introduces the Q function. Therefore, the optimal value function V^* and Q^* are the solutions of the presented LP. QREPS changes this LP with the same underlying idea as originally introduced in REPS. Therefore, p is constrained by the KL (Eq. (2)) and d is constrained by a "conditional relative entropy term"

$$H(d||d') = \sum_{x,a} d(x,a) \log \frac{\pi_d(a|x)}{\pi_{d'}(a|x)}.$$

Additionally, the constraints on the distributions p and d are relaxed to only require feature equivalence. π_0 and p_0 are introduced as reference distributions. With the usage of two parameters α, η we can formulate the primal problem of QREPS as

$$\begin{aligned} \underset{p,d}{\text{maximize}} \quad & \langle p, r\rangle - \frac{1}{\eta} D(p||p_0) - \frac{1}{\alpha} H(d||d_0) \\ \text{s.t.} \quad & E^T d = \gamma P^T p + (1-\gamma)\nu_0 \\ & \Phi^T d = \Phi^T p \qquad p \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}},\ d \in \mathbb{R}_+^{\mathcal{X}\times\mathcal{A}}. \end{aligned}$$

The Q function is defined to be parametrized by θ as $Q_\theta(x,a) = \langle\theta, \varphi(x,a)\rangle$ and we derive a value function from the Q function as

$$V_\theta(x) = \frac{1}{\alpha} \log \left(\sum_a \pi_0(x,a) e^{\alpha Q_\theta(x,a)} \right).$$

The bellman error function is introduced as

$$\Delta_\theta = r + \gamma \mathcal{P} V_\theta - Q_\theta,$$

which allows us to formulate the solutions p^*, ϕ_{d^*} from the value and Q function

$$\begin{aligned} p^*(x,a) &\propto p_0(x,a) e^{\eta \Delta_{\theta^*}(xa)} \\ \phi_{d^*}(a|x) &= \pi_0(a|x) e^{\alpha(Q_{\theta^*}(x,a) - V_{\theta^*}(x))}. \end{aligned}$$

This leaves us with the problem of getting the optimal parameters θ^*. Bas-Serrano et al. [8] show that this can be achieved by minimizing the convex function named Logistic Bellman Error (LBE)

$$\mathcal{G}(\theta) = \frac{1}{\eta} \log \left(\sum_{x,a} p_0(x,a) e^{\eta \Delta_\theta(x,a)} \right) + (1-\gamma)\langle\nu_0, V_\theta\rangle.$$

As this is impossible in a concrete implementation we need to introduce the empirical versions for both the bellman error Δ_θ and the LBE $\mathcal{G}$ for a batch of sample transitions $\{\xi_n\}_{n=1}^N$

$$\begin{aligned} \hat{\Delta}_\theta(x,a,x') &= r(x,a) + \gamma V_\theta(x') - Q_\theta(x,a) \\ \hat{\mathcal{G}}(\theta) &= \frac{1}{\eta} \log \left(\frac{1}{N} \sum_{n=1}^N e^{\eta \hat{\Delta}(\xi_n)} \right). \end{aligned}$$

Algorithm 1: MinMax-Q-REPS from [8].

Initialize π_0 arbitrarily;
for $k = 0, 1, \ldots, K-1$ **do**
  Run π_k and collect sample transitions $\{\xi_{k,n}\}_{n=1}^N$;
  Saddle-point optimisation for **Q-REPS-Eval**:
  for $\tau = 1, 2, \ldots, T$ **do**
    $\theta_{k,\tau} \leftarrow \theta_{k,\tau-1} - \beta \hat{g}_{k,\tau-1}(\theta)$;
    $z_{k,\tau}(n) \leftarrow \frac{(z_{k,\tau-1}(n) \exp(\beta' h_{k,\tau-1}(n))}{\sum_m (z_{k,\tau-1}(m) \exp(\beta' h_{k,\tau-1}(m)))}$;
  end
  Average weights over optimisation steps:
  $\theta_k = \frac{1}{T} \sum_{\tau=1}^T \theta_{k,\tau}$;
  Policy Update:
  $\pi_{k+1}(a|x) \propto \pi_0(a|x) e^{\alpha \sum_k Q_{\theta_k}(x,a)}$;
end
Result: π_I with $I \sim \mathrm{Unif}(K)$

This biased estimator of the LBE, named Empirical Logistic Bellman Error (ELBE), can be optimised for the optimal values θ^*. Similar to the problem in REPS, this is due to the linear operator P^T. But in QREPS, the bias of this estimator can be controlled by changing the value of η and with small enough values for η the convergence can be guaranteed [8].

A. Practical Implementation

For the optimisation of the ELBE [8] proposes a practical implementation, shown in Algorithm 1, named MinMax-Q-REPS. Their proposal consists of a two-player game between a sampler and a learner which try to do mirror descent on the function

$$\begin{aligned} \mathcal{S}(\theta, z) = & \sum_n z(n) \left(\hat{\Delta}_\theta(\xi_n) - \frac{1}{\eta} \log\left(N z(n)\right) \right) \\ & + (1-\gamma)\langle\nu_0, V_\theta\rangle, \end{aligned}$$

which is equivalent to optimising the ELBE. The only newly introduced variable is the distribution z which is a probability distribution over the number of transitions $[N]$. The learner now samples an index I from the distribution z and takes the corresponding transition $\xi_I = (X_I, A_I, X_I')$. Additionally, they sample a state $\bar{X}$ from the initial distribution ν_0. They now use the policy $\pi_\theta(a|x) = \pi(a|x) e^{\alpha Q_\theta(x,a) - V_\theta(x)}$ to sample two actions $A' \sim \pi_\theta(\cdot|X')$ and $\bar{A} \sim \pi_\theta(\cdot|\bar{X})$. The gradient is estimated by

$$\hat{g}(\theta) = \gamma\varphi(X', A') - \varphi(X, A) + (1-\gamma)\varphi(\bar{X}, \bar{A})$$

and the parameters θ are updated using Stochastic Gradient Descent (SGD) [11] or Adam [12] depending on the problem.

After every step of the learner the sampler updates the distribution z. For updating the distribution many ways are possible. The paper presents exponentiated gradient (EG) and

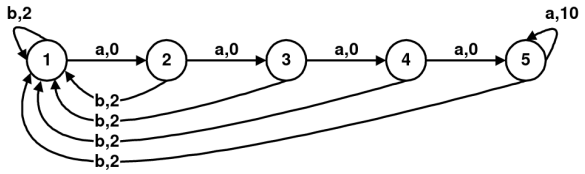

Figure 1: Schematics for the NChain environment with five states from [14].

best response (BR). EG uses a step size parameter β' to update the distribution in every iteration τ as follows

$$h_\tau(n) = \hat{\Delta}_\theta(\xi_n) - \frac{1}{\eta}\log\left(N z_\tau(n)\right)$$
$$z_{\tau+1}(n) \propto z_\tau(n) e^{\beta' h_\tau(n)}.$$

The BR update, defined as

$$z_{\tau+1}(n) \propto e^{\eta \hat{\Delta}_\tau(\xi_n)},$$

is overly aggressive according to the experiments of [8], but can be useful for problems such as cartpole (explained in Section IV-B2).

After optimising the Q function, the policy needs to be updated to reflect the latest changes to keep the convergence guarantee. This is done for the policy π_{k+1} in iteration $k+1$ as

$$\pi_{k+1}(a|x) \propto \pi_k e^{\alpha Q_k(x,a)}.$$

Noticeably, in [8] the authors mention in the conclusion that they need to save all Q functions to perform the update of the policy, due to them not assuming a parametric policy. As we are using a parametric policy we can use the formulation

$$\pi_{k+1}(a|x) \propto \pi_k(a|x) e^{\alpha Q_k(x,a)},$$

which is equivalent to the formulation in the algorithm

$$\pi_{k+1}(a|x) \propto \pi_0(a|x) e^{\alpha \sum_{i=0}^{k} Q_{\theta_i}(x,a)},$$

to update our policy. This allows us to not store the past Q functions.

IV. Implementation

This chapter summarises all important aspects of the implementation of QREPS belonging to this paper. As a framework for the implementation, PyTorch [13] was chosen.

A. DeepMind Control

Reinforcement learning algorithms are implemented against a set of interfaces that define the interaction with the environment. There are multiple known interfaces that allow to use multiple environments. Two of these well known interfaces are OpenAI Gym [15] and dm_env [16]. We decided on implementing using the DM ecosystem. DM created dm_env to create a consistent scheme for all their implementations. Wrappers for the conversion from and to OpenAI Gym are available in both directions.

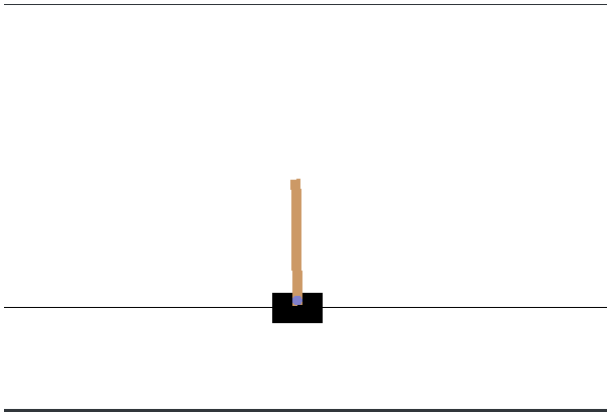

Figure 2: OpenAI Gym cartpole environment [15].

We decided for DM to more easily use the environments in the DM Control Suite [17]. Additionally, this allows us to use the DM behaviour suite for reinforcement learning [18] in future work to generate a behaviour analysis of REPS and QREPS.

B. Environments

1) NChain: The first environment for evaluating the implementation is NChain [14] using the implementation of [15]. NChain is a discrete state environment consisting of n linear states and two actions. The forward action a moves one step forward and the backward action b moves back to the beginning. When choosing the last action in the last state, the agent remains in the last state. The reward function awards a small reward for the backwards actions and a large reward for choosing the forward action in the last state of the chain. In Fig. 1 a visualization of the problem is shown.

One can configure a probability for the agent to "slip", in this case the agent executes the opposite action to the action chosen by the policy. We use the default values for NChain from OpenAI Gym as $n = 5$ and a slip probability of $p_{\text{slip}} = 0.2$.

2) Cartpole: The cartpole environment consists of a pole on top of a rolling cart. The goal is to balance the pole on top of the cart without it toppling over. The observation usually consists of the angle of the pole on the cart and the velocity of the cart. For every time step that the pole did not topple over the agent receives a fixed reward.

There are multiple different implementations for the cartpole environment. The easiest to solve is the implementation from OpenAI Gym [15]. In this implementation the action is discrete with the two possibilities left or right. A picture of this environment is shown in Fig. 2. A harder version of the environment is available in [17]. The cart is simulated using MuJuCo [19], therefore the problem is completely continuous and the action is the acceleration on the cart. Additionally, the computation time for the implementation in [17] is higher.

C. REPS

Although REPS can be implemented straight forward from the mathematical formulation, the described policy has some

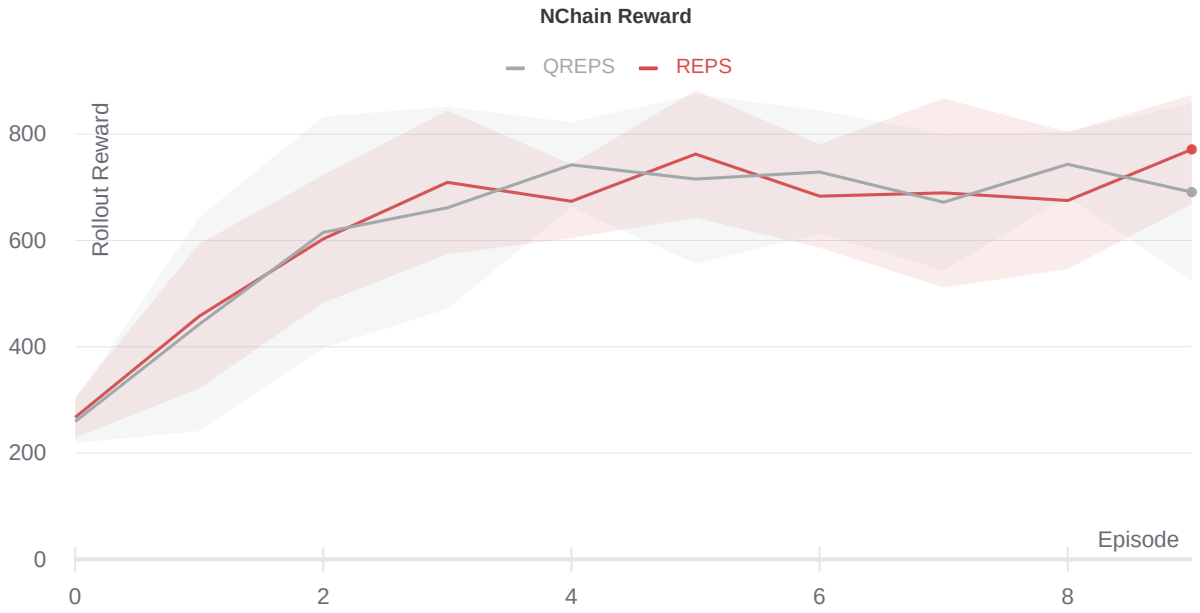

Figure 3: Reward per episode when running QREPS and REPS in NChain.

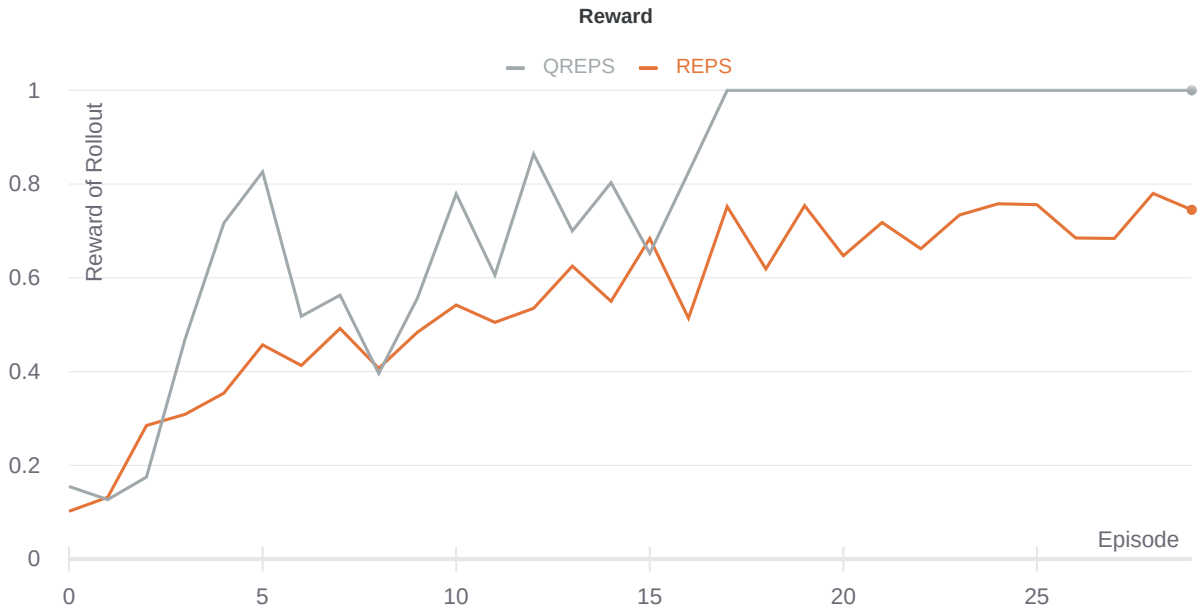

Figure 4: Reward per episode when running QREPS and REPS in Cartpole.

limitations. As usually only samples for the states and actions are available, the direct setting of the policy as described in Section II-B is not possible. A more elegant and useful implementations therefore optimises the weighted negative log-likelihood (NLL)

$$\text{weightedNLL} = \sum_{x,a} \log(\pi(a|x)) \exp\left(\frac{\delta_{\theta^*}(x,a)}{\eta*}\right)$$

instead of directly setting the probability. As this loss is differentiable with regard to the policy, it can be used to optimise the policy π with gradient descent. We use SGD [11] or Adam [12] to update the parameters of the policy.

The different possibilities for the constraining or regularization of the relative entropy as presented in Section II-B can be easily implemented by changes to the initialization of η and ε. The original implementation has a fixed ε and a learnable η. When we change η to be fixed and the value of ε to 0.0 we can implement the regularized entropy version without further changes. Due to the convex property of the dual function and the limited amount of parameters, we choose Limited-Memory BFGS (LBFGS) [20] as an optimiser for the dual.

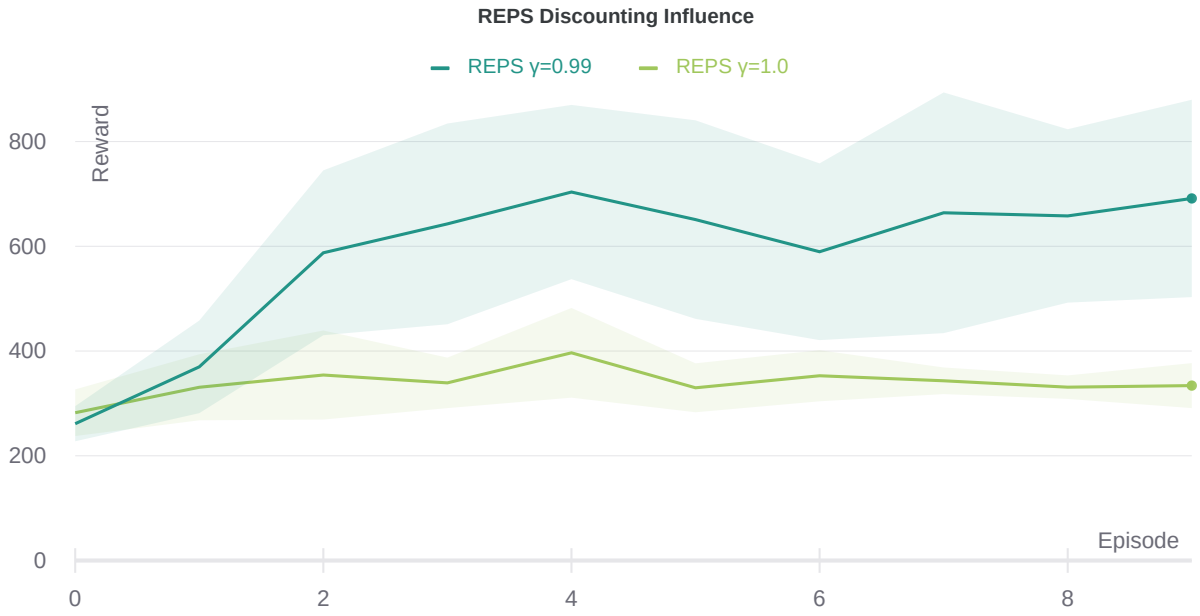

Figure 5: Reward per episode when running REPS on NChain with different discount factors γ.

D. Min-Max Q-REPS

To implement QREPS we follow the proposed Algorithm 1. The policy update is implemented similar to Section IV-C but with changes to the calculation of the NLL to use the Q function

$$\text{weightedNLL} = \sum_{x,a} \log(\pi(a|x)) \exp\left(\alpha Q^*(x,a)\right).$$

The gradient is estimated as outlined in Section III-A but with the possibility to use multiple samples and average between them.

V. Evaluation

We ran REPS and QREPS on both environments, which were presented in Section IV-B. This section focuses on the achieved results and compares the performance of REPS and QREPS.

A. Hyperparameter

To enable a good performance for both algorithms we performed a hyperparamter optimisation for every combination. For an efficient optimisation, WanDB's [21] sweep feature was used. It automatically runs the script multiple times and performs Bayesian optimisation on the hyperparamters [14]. An important parameter is the used feature function for each environment. For NChain tabular features were used which map the number of the state and action to a one-hot encoding. For Cartpole we used a 2-layer neural net with 200 neurons and ReLU as an activation function. The network uses the default PyTorch initilization and freezes the layers to prevent updates. These features are equivalent to the features used in [8]. The parameters for REPS and QREPS can be seen in Table I and Table II respectively.

B. NChain

The performance of both tuned algorithms can be seen in Fig. 3. We ran 10 runs for each algorithm and plot the average and standard deviation of the reward. Noticeably, QREPS performs similar but has more variance than the runs of [8]. Additionally, we used a smaller η and therefore would expect a smaller variance and better convergence. Another important aspect is our performance of REPS. In contrast to the original paper our implementation performs very similar to QREPS. This could be very well due to the used discount. As seen in Table I we used a discount which is smaller than 1.0. When only changing the discount $\gamma = 1.0$ REPS was not able to achieve any learning on the environment, which can be seen in Fig. 5. Our assumption, that Bas-Serrano et al. [8] used a discount $\gamma = 1.0$, could be verified when they released their code. We suppose when using REPS with $\gamma = 1.0$, the value function collapses and values every state equally. This could have been expected as outlined in Section II-A, when introducing the discount factor. The value function tries to include all possible rewards of the future and as each state potentially reaches all other states, the value function collapses. This does not happen with QREPS, due to the Q function differentiating between the action that leads to the next state and the action that returns to the beginning. In that case the forward action is valued better and the agent learns the correct policy. In our experiments QREPS does not perform much better than REPS.

C. Cartpole

In cartpole we compare single runs between REPS and QREPS, when running them for 30 episodes. The results of these runs can be seen in Fig. 4. The reward is normalized to $[0, 1]$, therefore 1.0 is the perfect reward, which is equivalent

Environment	η	dual LR	γ	T	Features
NChain	2.0	0.07	0.99	300	Tabular
Cartpole	5.0	0.01	1.0	300	2 Layer NN

Table I: Used hyperparameters for REPS.

Environment	η	β	β'	γ	T	Learner	Sampler	Features
NChain	5.0	0.05	0.1	1.0	300	SGD	EG	Tabular
Cartpole	4.8	0.02	-	0.99	300	Adam	BR	2 Layer NN

Table II: Used hyperparameters for QREPS.

to the agent keeping the pole steady for all timesteps. Just as in [8] QREPS performs better and achieves a perfect reward after around 20 episodes. Again, our REPS implementation seems to perform better than the original REPS implementation. Still, we are able to verify the functionality of QREPS and the usability of the algorithm for both environments.

VI. Conclusion

We were able to reproduce the findings of [8]. We successfully implemented QREPS according to their practical framework specification.

When running our implementation on cartpole and NChain we observed our REPS to perform better than the original implementation. Our assumption that the discount is responsible for this difference in performance was verified with the implementation of the original work.

Additionally, our implementation shows that the assumption of not needing to store the past Q functions as mentioned in Section III-A seems to hold. This could be verified further by reproducing all original environments.

Further work could try to generalize QREPS on more than linear Q functions and work on an implementation for continuous action spaces as these seem to be still limiting factors of the current formulation.

References

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, number: 7676 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature24270

[2] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," *arXiv:1912.06680 [cs, stat]*, Dec. 2019, arXiv: 1912.06680. [Online]. Available: http://arxiv.org/abs/1912.06680

[3] M. P. Deisenroth, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 1–142, 2011. [Online]. Available: http://www.nowpublishers.com/articles/foundations-and-trends-in-robotics/ROB-021

[4] Z. Zhang, S. Zohren, and S. Roberts, "Deep Reinforcement Learning for Trading," *The Journal of Financial Data Science*, vol. 2, no. 2, pp. 25–40, Apr. 2020. [Online]. Available: http://jfds.pm-research.com/lookup/doi/10.3905/jfds.2020.1.030

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement

[5] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, May 1992. [Online]. Available: http://link.springer.com/10.1007/BF00992698
Learning," *arXiv:1312.5602 [cs]*, Dec. 2013, arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602

[7] J. Peters, K. Mulling, and Y. Altun, "Relative Entropy Policy Search," p. 6, 2010.

[8] J. Bas-Serrano, S. Curi, A. Krause, and G. Neu, "Logistic Q-Learning," *arXiv:2010.11151 [cs, stat]*, Oct. 2020, arXiv: 2010.11151. [Online]. Available: http://arxiv.org/abs/2010.11151

[9] M. L. Puterman, "Chapter 8 Markov decision processes," in *Handbooks in Operations Research and Management Science*, ser. Stochastic Models. Elsevier, Jan. 1990, vol. 2, pp. 331–434. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0927050705801720

[10] G. Neu, A. Jonsson, and V. Gómez, "A unified view of entropy-regularized Markov decision processes," *arXiv:1705.07798 [cs, stat]*, May 2017, arXiv: 1705.07798. [Online]. Available: http://arxiv.org/abs/1705.07798

[11] H. Robbins, "A Stochastic Approximation Method," 2007.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[14] M. Strens, "A Bayesian Framework for Reinforcement Learning," p. 8.

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540 [cs]*, Jun. 2016, arXiv: 1606.01540. [Online]. Available: http://arxiv.org/abs/1606.01540

[16] Alistair Muldal, Yotam Doron, John Aslanides, Tim Harley, Tom Ward, and Siqi Liu, "dm_env: A Python interface for reinforcement learning environments," 2019, original-date: 2019-07-08T13:58:50Z. [Online]. Available: https://github.com/deepmind/dm_env

[17] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller, "DeepMind Control Suite," *arXiv:1801.00690 [cs]*, Jan. 2018, arXiv: 1801.00690. [Online]. Available: http://arxiv.org/abs/1801.00690

[18] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvari, S. Singh, B. Van Roy, R. Sutton, D. Silver, and H. Van Hasselt, "Behaviour Suite for Reinforcement Learning," *arXiv:1908.03568 [cs, stat]*, Feb. 2020, arXiv: 1908.03568. [Online]. Available: http://arxiv.org/abs/1908.03568

[19] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 5026–5033, iSSN: 2153-0866.

[20] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, Aug. 1989. [Online]. Available: https://doi.org/10.1007/BF01589116

[21] "Weights & Biases." [Online]. Available: https://wandb.ai/