Entropy Regularized Reinforcement Learning with Cascading Networks

Riccardo Della Vecchia* riccardo.della-vecchia@inria.fr
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL
Lille, France

Alena Shilova* alena.shilova@inria.fr
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL
Lille, France

Philippe Preux philippe.preux@inria.fr
Inria, Univ. Lille, CNRS
Lille, France

Riad Akrour* riad.akrour@inria.fr
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 - CRIStAL
Lille, France

Abstract

Deep Reinforcement Learning (Deep RL) has had incredible achievements on high dimensional problems, yet its learning process remains unstable even on the simplest tasks. Deep RL uses neural networks as function approximators. These neural models are largely inspired by developments in the (un)supervised machine learning community. Compared to these learning frameworks, one of the major difficulties of RL is the absence of i.i.d. data. One way to cope with this difficulty is to control the rate of change of the policy at every iteration. In this work, we challenge the common practices of the (un)supervised learning community of using a fixed neural architecture, by having a neural model that grows in size at each policy update. This allows a closed form entropy regularized policy update, which leads to a better control of the rate of change of the policy at each iteration and help cope with the non i.i.d. nature of RL. Initial experiments on classical RL benchmarks show promising results with remarkable convergence on some RL tasks when compared to other deep RL baselines, while exhibiting limitations on others.

Keywords: Policy iteration, Entropy regularization, Cascade neural networks, Mirror descent

1. Introduction

Reinforcement learning (RL) formulates a general machine learning problem in which an agent has to take a sequence of decisions to maximize a supervision signal (Sutton and Barto, 2018; Szepesvari, 2010). RL as a learning framework, especially when combined with neural function approximators, has made large practical breakthroughs over the last years on complex, high dimensional problems (Silver et al., 2016; Mnih et al., 2013), and its range of application continues to grow to new domains such as drug discovery (Bengio et al., 2021). Surprisingly however, even simple RL tasks can exhibit one of the main disadvantages of current deep RL algorithms: their training is unstable and does not exhibit convergence to an optimal policy. Rather, deep RL discovers good policies along the way with often large performance oscillations in-between.

So what makes the training process of a neural model by RL more brittle than say, the training of the same model to regress a continuous signal or classify images? Certainly, one of the major challenges of RL, and specifically online RL[1], is that the agent's decisions influence the data gathering process, violating the typical assumption of the aforementioned supervised learning tasks that data is independent and identically distributed (Bishop, 2006). To

*. These authors contributed equally to this work.

1. We note that offline, batch, RL also violates the independent and identically distributed assumption since there is still a mismatch between the distribution of training data and the data generated by the learned policy, which is inherent to the sequential nature of RL in general. However, this work will only focus on the online RL setting.

cope with this challenge, several RL algorithms constrain the agent's behavior to only slowly change. In trajectory optimization and optimal control, a new policy is made close to the policy around which the dynamics of the system have been approximated through mixing (Todorov and L., 2005; Tassa et al., 2014) or by a Kullback-Leibler (KL) constraint (Levine and Abbeel, 2014). In policy gradient, a key breakthrough was the use of natural gradient that follows the steepest descent in policy space rather than parameter space (Bagnell and Schneider, 2003; Peters and Schaal, 2008; Bhatnagar et al., 2009), i.e. seeking maximal objective improvement with minimal *policy* change.

Constraining successive policies to be close to each others in approximate policy iteration is justified in the seminal work of Kakade and Langford 2002 by the mismatch between what the policy update should optimize (advantage function in expectation of its own state distribution) and what is optimized in practice (advantage function in expectation of the state distribution of the data gathering policy). As in optimal control, closeness can be achieved by mixing policies (Kakade and Langford, 2002; Pirotta et al., 2013), limiting deviation of their probability ratio to one (Schulman et al.) or constraining their KL-divergence (Schulman et al., 2015; Abdolmaleki et al., 2018; Tangkaratt et al., 2018; Akrour et al., 2018). In the latter case, when KL-divergence is used to regularize the policy update, a recent line of research has drawn similarities between these algorithms and the convex optimizer mirror descent (Neu et al., 2017; Geist et al., 2019; Abbasi-Yadkori et al., 2019), which provides as a result convergence guaranties to an optimal policy. Of course, practical implementations of these algorithms may still fail to find the optimal policy due to two difficulties: **i)** learning the Q-function of the current policy from sample data and **ii)** solving the entropy regularized policy update in the neural parameter space.

In this paper, we investigate the use of Cascade Neural Networks (Cascade-NN, (Fahlman and Lebiere, 1989)) in the context of entropy regularized policy iteration to address **ii**. Unlike typical neural models, Cascade-NNs can grow in size during learning. Importantly, when new neurones are added, old ones are frozen which allows us to perform the policy update in closed form and completely eliminate the second source of errors discussed above. In the remainder of the paper, we will first describe in a preliminaries section Cascade-NNs and entropy regularized RL (Section 2), before describing our approach in Section 3. We discuss related work in Section 5 after comparing our algorithm in Section 4 to deep RL baselines on classical RL benchmarks.

2. Preliminaries

In this section, we introduce our notations and briefly describe the framework of entropy regularized RL. We then present the general architecture of Cascade-NNs as described in the seminal paper of (Fahlman and Lebiere, 1989) before describing how it is used in our algorithm.

2.1 Notation

We consider Markov Decision Problems (MDPs) defined as a tuple $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$ to model the interactions of the agent with the environment. $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, r is the unknown reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and P is the unknown probability transition function $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$, where $\Delta_{\mathcal{S}}$ is the set of distributions over $\mathcal{S}$.

Let the policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ be a mapping between a state to a distribution over actions. For every such policy π, one can compute the value function

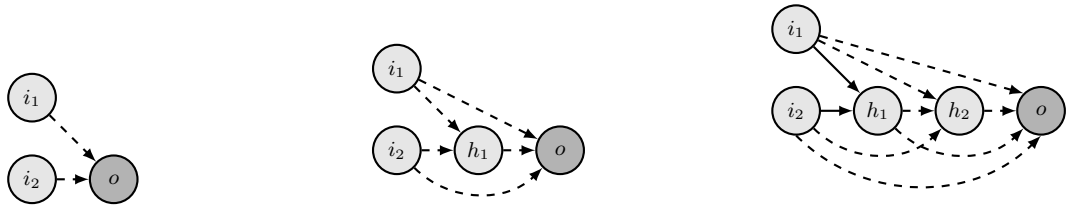
$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \;\middle|\; s_0 = s\right],$$

where the expectation is taken w.r.t. all states and actions following s. We also define the state-action value function as

$$Q_\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(s,a)} V_\pi(s').$$

2.2 Entropy regularized policy iteration

RL considers the problem of finding the optimal policy in the MDP's unknown environment. One way of doing so is to use policy iteration methods that successively perform **i)** a policy evaluation step to compute Q_π and **ii)** a policy



(a) No hidden layers (b) 1 hidden layer (c) 2 hidden layers

Figure 1: Incremental growth in Cascade-NN architecture and freezing of weights. Dashed lines show learnable parameters after each addition, while solid lines represent frozen weights. In a Cascade-NN, when new hidden nodes are added, all inputs to older nodes are frozen so as to freeze older features.

improvement step, yielding a new policy π' that picks at every state s an action in $\arg\max_a Q_\pi(s,a)$. Repeatedly performing these two steps will converge in finite time to an optimal policy, irrespective of the choice of the initial policy (Sutton and Barto, 2018). However, we notice that at any iteration, the policy π' obtained after a policy improvement step does not explore actions that do not maximize the previous Q-function $Q_\pi(s,.)$, for every state s. Hence, to correctly estimate $Q_{\pi'}(s,.)$ for such actions in practice, one would need to introduce another data gathering policy that is more explorative than π'. An alternative is to regularize the policy update step to maintain the stochasticity of π'. This is usually performed by adding a KL-divergence term between π and π', ensuring that exploration does not vanish too quickly.

In contrast to these algorithms, in this paper, we investigate solving the soft policy update exactly for every state, using an incrementally increasing neural architecture. Given $\hat{Q}_{\pi_i}$, the approximation of the Q-function of π_i at iteration i of the algorithm, our policy update at every state s is similar to that of MPO (Abdolmaleki et al., 2018) and POLITEX (Abbasi-Yadkori et al., 2019) and is given by the following optimization problem:

$$\begin{aligned}\pi_{i+1}(s) &= \arg\max_\pi \mathbb{E}_{a\sim\pi(.|s)}\hat{Q}_{\pi_i}(s,a) - \frac{1}{\eta}\mathrm{KL}(\pi(s)\|\pi_i(s)),\\ &\propto \exp\left(\eta\hat{Q}_{\pi_i}(s,\cdot)\right)\pi_i(.|s),\end{aligned} \tag{1}$$

where $\mathrm{KL}(\pi(s)\|\pi_i(s))$ is the KL-divergence between the distributions over $\mathcal{A}$ that are $\pi(s)$ and $\pi_i(s)$ at state s. By recursion and assuming that π_0 is the uniform distribution over $\mathcal{A}$, we get

$$\pi_{i+1}(s) \propto \exp\left(\eta\sum_{k=0}^{i}\hat{Q}_{\pi_k}(s,\cdot)\right). \tag{2}$$

We thus see that the entropy regularized policy update can be solved in closed form, and policies have a very simple form provided we can keep track of all previously estimated Q-functions. Of course, this might appear too cumbersome in practice if we consider that each Q-function is a separate neural network. For this reason, other papers approximated the update, e.g. in MPO (Abdolmaleki et al., 2018), by fitting a Gaussian neural policy minimizing the divergence to $\exp\left(\eta\hat{Q}_{\pi_i}(s,\cdot)\right)\pi_i(.|s)$. Alternatively, in (Lazic et al., 2021), authors have considered keeping only the last few Q-functions or keeping a large replay memory of past MDP transitions, and fitting a single network to the sum of Q-functions. In contrast, our policy will take the exact form of Eq. (2). However, we will not train independent neural networks for each $\hat{Q}_{\pi_k}$, but leverage the Cascade-NN architecture and only add a few neurons at every iteration. The rationale is that since the policy will not change drastically between iterations neither will their Q-function and one might need only a few more neurons to learn $\delta_k = \hat{Q}_{\pi_k} - \hat{Q}_{\pi_{k-1}}$.

2.3 Cascade-NN

Cascade Neural Network (Cascade-NN) or also called Cascade-Correlation Networks (Fahlman and Lebiere, 1989) is a special type of neural network architecture that is growing at each epoch by n (typically $n = 1$) new neurons that are connected to the input of the NN and all previously created hidden neurons. At first, it has no hidden layer

Algorithm 1 Pseudo code of the MICARL Algortihm

Set $\hat{Q}_{\pi_0}$ to the zero function
for Iteration i in $\{1, \dots, \text{NB\_ITER}\}$ **do**
  Collect NB_SAMP transitions of type (s, a, r, s', a') from the environment following policy $\pi_i(s) \propto \exp\left(\eta \sum_{k=0}^{i-1} \hat{Q}_{\pi_k}(s, \cdot)\right)$
  Add n neurons to the current Cascade-NN
  Set δ^0 to the zero function
  for Epoch e in $\{1, \dots, E\}$ **do**
    Compute target $r + \gamma(\hat{Q}_{\pi_{k-1}} + \delta^{e-1})(s', a') - \hat{Q}_{\pi_{k-1}}(s, a)$ for every transition (s, a, r, s', a')
    Obtain δ^e by fitting the target using stochastic gradient descent
  end for
  Set $\hat{Q}_{\pi_k} = \hat{Q}_{\pi_{k-1}} + \delta^E$
end for

(Figure 1a), and gradually more and more neurons get added to its structure (Figures 1b and 1c). Historically, these newly added neurons were trained following a three phase process. Firstly, a batch of neurons larger then n is added and the output of each of those new neurons is trained to maximize the correlation with the current residual error of the model. Secondly, the neurons are ranked according to their correlation and only the n-top neurons are kept for the final phase. Thirdly, new neurons are connected to the output layer and the weights of the output layer are updated in order to minimize the residual error. Importantly, throughout all phases older neurons are frozen, which means that one can easily compute the sum of all past outputs $\sum_k o_k$ by simply summing all past weights of the (linear) output layer. An illustration of the freezing procedure is given in Figure 1, showing all re-trainable parameters at every iteration. Here i_1 and i_2 correspond to data inputs, h_1 and h_2 to two hidden neurons and o to the output. Solid edges show the frozen connections, while dashed edges are for trainable parameters.

In (Fahlman and Lebiere, 1989), the authors emphasize the following advantages of Cascade-NN with respect to classical MLP neural networks:

- **Non-parametric training**, there might be less hyper-parameters to tune such as the depth, width and connectivity of NN, its training (and growth) is stopped automatically as soon as a stopping criterion is satisfied;
- **Fast learning**, freezing all the layers except the last one helps to optimize the parameters without doing back-propagation, in this way all neurons have their independent goal and they can "settle into distinct useful roles";
- **Incremental learning**, especially useful when the model constantly receives some new information (data) in a stream manner, in this case old features are preserved, while new features enrich the feature extraction for the newly obtained data.

Prior work already explored the use of Cascade-NN in RL. Notably, (Girgin and Preux, 2008) combined features trained to maximize correlation with the Bellman residual before using LSPI (Lagoudakis and Parr, 2003) to find an optimal policy given the current set of features. However, (Girgin and Preux, 2008) did not leverage the properties of Cascade-NN to perform entropy regularized policy update in closed form and investigate the stability of this approach in a deep RL context, which is the main contribution of this paper. Regarding the learning of the Q-function, we do not use LSTDQ (Lagoudakis and Parr, 2003). Instead, we implemented a more streamlined training procedure using a (neural) fitted Q-iteration scheme, similar to DQN (Mnih et al., 2015)—although using the Bellman operator of the current policy instead of the Bellman optimality operator in DQN—for simultaneously training the features and learning the Q-function of the current policy. However, we discuss how Cascade-NN techniques presented here can be used in the context of our work at the end of Section 3.

3. The MICARL algorithm

In this section, we introduce our algorithm MICARL. MICARL is a deep RL algorithm that follows the policy iteration scheme. To approximate Q_{π_k}, we use the adapted Cascade-NN model depicted in Fig. 2. This neural model has two

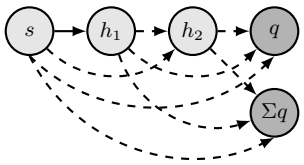


Figure 2: Our Q-function and policy network. The Cascade-NN has two heads, one storing the last Q-function—used to compute the targets of the neural fitted Q step in Alg. 1, while the output Σq accumulates all past weight matrices of node q, and is used by the softmax policy. Note that all nodes in the figure can be multi-dimensional, including hidden nodes/layers. See the implementation details for more information.

heads giving the last Q-function and the sum of the last Q-functions. Alg. 1 shows the pseudo-code for MICARL. Starting with a zero function $\hat{Q}_{\pi_0}$, at every iteration $i > 0$ we collect a dataset $\mathcal{D}^{\pi_i}$ of transition samples of type (s, a, r, s', a'), where actions are sampled from the current policy $\pi_i \propto \exp(\eta \sum_{k=0}^{i-1} \hat{Q}_{\pi_k})$ following the entropy regularized policy update of Eq. (2), and the next state s' and rewards are given by the environment. Using the transition dataset $\mathcal{D}^{\pi_i}$, we learn the Q-function approximation $\hat{Q}_{\pi_k}$ following a standard (neural) fitted Q-iteration learning scheme. In our implementation, we learn $\delta_i = Q_{\pi_i} - Q_{\pi_{i-1}}$, instead of Q_{π_i}. Starting with a zero function δ_i^0, at each epoch e, we compute the targets $T_i^e(s, a)$ for every $(s, a, r, s', a') \in \mathcal{D}^{\pi_i}$,

$$T_i^e(s, a) = r + \gamma(\hat{Q}_{\pi_{i-1}} + \delta_i^{e-1})(s', a') - \hat{Q}_{\pi_{i-1}}(s, a). \tag{3}$$

We then update the learnable weights $\xi^{(i)}$and $W_\delta^{(i)}$ (see implementation details below) of the Cascade-NN for one epoch on the dataset $\mathcal{D}^{\pi_i}$ by minimizing the loss

$$\min_{\xi^{(i)}, W_\delta^{(i)}} \sum_{(s,a,r,s',a') \sim \mathcal{D}^{\pi_i}} \left(\delta_i^e(s, a) - T_i^e(s, a)\right)^2, \tag{4}$$

using stochastic gradient descent. Finally, the process of computing targets and fitting δ_i^e is repeated until reaching a given number of epochs E.

As for the policy update, we simply add the new Q-function $Q_{\pi_i} = Q_{\pi_{i-1}} + \delta_i^E$ to the head of the Cascade-NN accumulating all past Q-functions, completing the definition of π_{i+1}. Our implementation of MICARL, including all experimental results, are obtained using an on-policy setting. We note however that the extension of our algorithm to the off-policy setting is trivial (one simply needs to replace a' in a transition sample to match the current policy), but its investigation is left for future work.

Implementation details. Our Cascade-NN at any iteration i is taking state $s \in \mathcal{S}$ as an input and outputs either a vector $\sum_{k=0}^{i-1} \hat{Q}_{\pi_k}(s, \cdot)$ or a vector $\hat{Q}_{\pi_{i-1}}(s, \cdot)$. For a given Cascade-NN, we start with zero hidden neuron and we grow the architecture at each iteration $i \in \{1, \ldots, \text{NB\_ITER}\}$. In particular, at the beginning of iteration i we have $(i-1)n$ hidden neurons and during the iteration we grow it by n to reach a total of in neurons at the end of i-th iteration. Those hidden neurons are used to extract features. Further, what we consider as features is the direct inputs of output layers. At the beginning of the first iteration, the input is connected directly to the output layers, thus the input acts as the feature before the training. At any iteration, the input stays connected to the output layers and thus is a part of feature vectors. At the beginning of iteration i, its $(i-1)n$ old hidden neurons plus the input vector represent the feature vector function $\phi^{(i-1)} : \mathcal{S} \to \mathbb{R}^{(i-1)n+\dim \mathcal{S}}$ that for any state $s \in \mathcal{S}$ returns its corresponding feature vector $\phi^{(i-1)}(s)$ from iteration $i-1$. Further, those neurons are frozen and not trained. Moreover, we have access to the two output layers described above. The weight matrix $W^{(i-1)} \in \mathbb{R}^{|\mathcal{A}| \times ((i-1)n+\dim \mathcal{S})}$ corresponds to the sum of Q-values so that the matrix-vector product $W^{(i-1)}\phi^{(i-1)}(s)$ gives the vector $\sum_{k=0}^{i-1} \hat{Q}_{\pi_k}(s, \cdot)$ of size $|\mathcal{A}|$ (for simplicity, we omit bias term). Therefore, this output layer can be used to compute values necessary for policy π_i from Eq. (2). The weight matrix $W_Q^{(i-1)} \in \mathbb{R}^{|\mathcal{A}| \times ((i-1)n+\dim \mathcal{S})}$ corresponds to the approximation Q-value of policy π_{i-1}, that is $\hat{Q}_{\pi_{i-1}}(s, \cdot) = W_Q^{(i-1)}\phi^{(i-1)}(s)$. During iteration i, n new neurons are generated in a cascade manner as described in Section 2.3. They are responsible for computing a new component of the features, in particular they correspond to a vector function $\tilde{\phi}_{\xi^{(i)}}^{(i)} : \mathbb{R}^{n(i-1)+\dim \mathcal{S}} \to \mathbb{R}^n$ that takes features from the previous

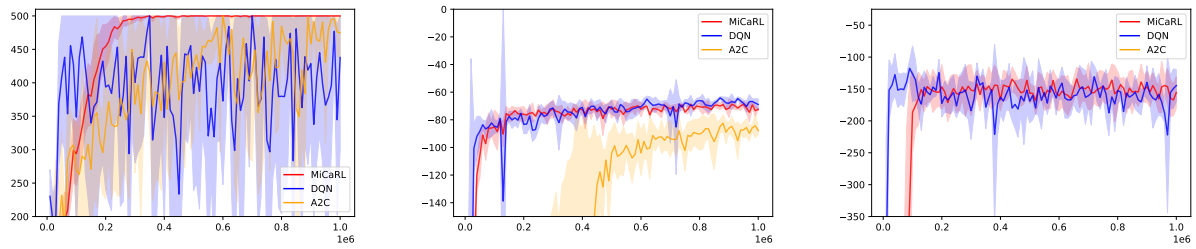


Figure 3: Zoom of the reward curves around their asymptotic values on CartPole-v1 (left), Acrobot-v1 (center), and Pendulum-v1 (right).

iteration $\phi^{(i-1)}(s)$ as an input and outputs $\tilde{\phi}^{(i)}_{\xi^{(i)}}(\phi^{(i-1)}(s))$, which is further concatenated[2] with $\phi^{(i-1)}(s)$ to constitute $\phi^{(i)}(s) = \text{CAT}\left(\phi^{(i-1)}(s), \tilde{\phi}^{(i)}_{\xi^{(i)}}(\phi^{(i-1)}(s))\right) \in \mathbb{R}^{ni+\dim \mathcal{S}}$. Moreover, the new output layer is initialized with a weight matrix $W^{(i)}_{\delta} \in \mathbb{R}^{|\mathcal{A}|\times(ni+\dim \mathcal{S})}$ that should represent $\delta_i(s,\cdot) = \hat{Q}_{\pi_i}(s,\cdot) - \hat{Q}_{\pi_{i-1}}(s,\cdot) = W^{(i)}_{\delta}\phi^{(i)}(s)$. Parameters $\xi^{(i)}$ and $W^{(i)}_{\delta}$ are optimized to minimize the loss defined in Eq. (4). At the end of iteration i, once $\xi^{(i)}$ and $W^{(i)}_{\delta}$ are optimized, $\hat{Q}_{\pi_i}$ can be evaluated with $\hat{Q}_{\pi_{i-1}}(s,\cdot) + \delta_i(s,\cdot) = W^{(i-1)}_{Q}\phi^{(i-1)}(s) + W^{(i)}_{\delta}\phi^{(i)}(s)$, therefore by setting $W^{(i)}_{Q} = \text{CAT}(W^{(i-1)}_{Q}, O^{|\mathcal{A}|\times n}) + W^{(i)}_{\delta}$, where $O^{|\mathcal{A}|\times n}$ is a zero matrix of dimension $|\mathcal{A}| \times n$, the approximation $\hat{Q}_{\pi_i}(s,\cdot)$ is naturally obtained from $W^{(i)}_{Q}\phi^{(i)}(s)$. Similarly, $\sum_{k=0}^{i} \hat{Q}_{\pi_k}(s,\cdot) = W^{(i)}\phi^{(i)}(s)$ where $W^{(i)} = \text{CAT}(W^{(i-1)}, O^{|\mathcal{A}|\times n}) + W^{(i)}_{Q}$.

In our implementation, the total number of weights grows as $\mathcal{O}(i^2)$ since new neurons connect to all previous neurons. This might be too prohibitive in practice, and we will experiment in future work with variants where new neurons only connect to a fixed number of past neurons. For example, following the correlation ideas of the original Cascade-NN (see Section 2.3), one might select the most promising past neurons according to the correlation between their activation and the current Bellman residual.

4. Experiments

We evaluate MICARL on four different gym environments: CartPole-v1, Acrobot-v1, a discrete action-space version of Pendulum-v1 and MountainCar-v0 and we compare it with A2C and DQN agents as implemented in `rlberry` Domingues et al. (2021). In our experiments we use the default implementations of the agents: for the value function and the policy network in A2C, and for the Q-value function network of DQN, we take multilayer percecptrons with two hidden layers and 64 hidden units each. Our experiments show that MICARL is in general more stable than both A2C and DQN and is achieving performances superior or comparable to DQN on three out of four environments, as it is evident from Figures 3 and 4. We plot curves averaged over five different seeds, using the default seeding handler from (Domingues et al., 2021), and the shaded area indicates one standard deviation. Figure 3 is a zoom of the reward curves near the asymptotic values reached by the best performing algorithms. For this reason, for Pendulum-v1 we do not see the curve corresponding to A2C, since its performance is quite poor on this environment. In Figure 4, we plot a more detailed study in which we do not just show rewards but also losses during training, average normalized entropy (across visited states) and the KL-divergence between two consecutive policies for A2C and MICARL (not for DQN since it is not a policy iteration algorithm). Note that the way the loss is computed differs from one algorithm to another: MICARL and DQN compute Bellman residuals of different Bellman operators, while A2C tries to minimize the difference between critic output and Monte Carlo estimate of the value function based on the dataset. This explains different levels of loss values in loss plots.

2. We use CAT to denote concatenation with respect to the last mode of a tensor, so in case of matrices $A \in \mathbb{R}^{n\times k}$, $B \in \mathbb{R}^{n\times m}$, then $C = \text{CAT}(A, B) \in \mathbb{R}^{n\times(k+m)}$ is the matrix with first k columns from A and last m columns from B.

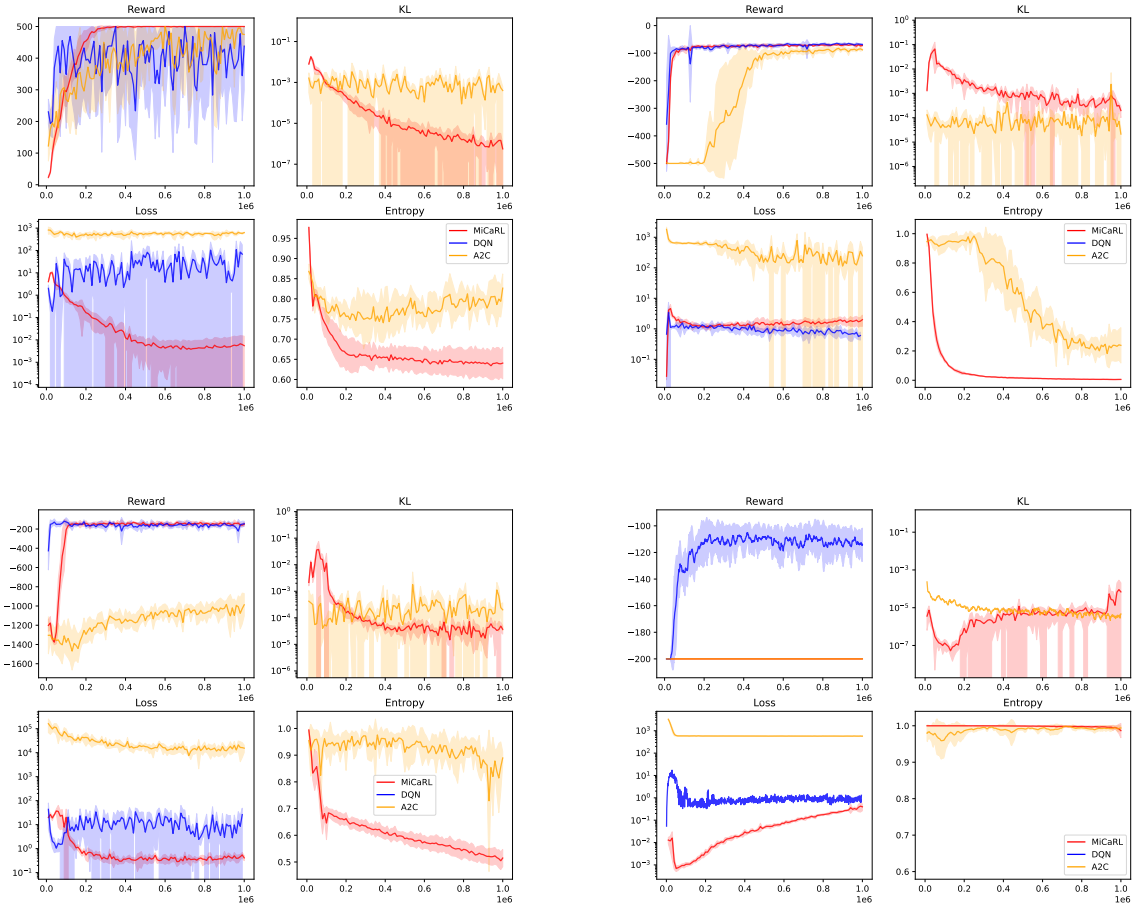


Figure 4: MICARL and baselines on CartPole-v1 (top-left corner), Acrobot-v1 (top-right corner), Pendulum-v1 (bottom-left corner) and MountainCar-v0 (bottom-right corner).

We note that the number of neurons n to add at every iteration is important. For small n, we typically lose the good properties of over-parameterization when learning δ. Especially, since in our implementation we do not use the Cascade-NN idea discussed in Section 2.3 of training a number of candidate neurons $> n$ before picking the n most promising ones. As a result, we found that using small n, e.g. $n = 1$, would often lead to dead neurons and more generally, to attraction to poor local optima. We address this issue by using a larger than usual n, typically $n \in \{10, 20, 50\}$. We try also combinations of the other hyper-parameters: the strength parameter of the KL-regularizer in Eq. (1) is $\eta \in \{0.01, 0.1, 0.5, 1, 5\}$ and the number of epochs per iteration $E \in \{64, 128\}$. Hyper-parameter search is done in grid search manner with the help of Ray Tune software Liaw et al. (2018). Results are quite similar for different choices of epochs, therefore we further keep it fixed $E = 64$. In contrast, our algorithm is more sensitive to the choice of η and slightly less to the choice of n. Further, we report separately for each environment the best choice of n and η. In addition, each trial is conducted with a number of iterations equal to NB_ITER $= 100$, batch-size $= 64$ and in each iteration we take NB_SAMP $= 10\,000$ steps in the environment. Therefore, the total number of steps in the environment is equal to $1\,000\,000$ for each experiment, which is the quantity on the x-axis in Figures 4 and 3. The final architecture of MICARL contains NB_ITER $\times\, n$ hidden neurons.

The strong stability of MICARL is probably due to the second term in Eq. (1) that acts as a regularizer. Nonetheless, MICARL achieves impressive and surprising performances. For example on CartPole-v1, MICARL is able to find a much better approximation of the Q-value function with respect to our baselines and the reward remains stable at its

maximum value after a brief training phase. On Acrobot-v1 and Pendulum-v1, MICARL is performing as good, or better, than DQN, and significantly better than A2C. MountainCar-v0 is instead known to be a difficult environment and MICARL is not able to learn anything, the same for A2C. In contrast, DQN has a good performance on this environment as well.

CartPole-v1 We test the performance of MICARL against the baselines of A2C and DQN on CartPole-v1 in Figure 4. The best performance of MICARL is observed for $n = 10$ and $\eta = 0.1$. The training loss of MICARL is getting closer to zero than the losses of the two other algorithms, showing that MICARL succeeds in better approximating Q-value functions. Furthermore, A2C and DQN loss curves are plateauing and oscillating, proving them to be unstable. The curves of the rewards for A2C and DQN also exhibit an oscillating behaviour. Sometimes they manage to reach the optimum value, which is 500 for this environment, but on average they do not manage to maintain this performance in all episodes. The stability of MICARL is also confirmed by the KL-divergence plot where at the end of the training the difference between successive policies becomes increasingly negligible, meaning that the algorithm is converging to a policy. Interestingly, the normalized entropy (having value between 0 for a deterministic policy and 1 for the uniform policy) of the policy is staying quite high, around 0.65, for MICARL. This reflects the fact that under a near optimal policy that balances the pole correctly, there are a lot of states where both actions are viable, meaning that our algorithm not only finds an optimal policy, but several optimal ones. A2C exhibits an even higher entropy, but since its reward is still oscillating, one cannot draw the same conclusions. Overall, MICARL achieves the best results, while being more stable than other baselines.

Acrobot-v1 and Pendulum-v1 In the case of Acrobot-v1, MICARL performs the best with $n = 50$, $\eta = 1$, while in case of Pendulum-v1 with $n = 50$ and $\eta = 0.1$. Results on Acrobot-v1 and a discrete version of Pendulum-v1 are almost identical to DQN in terms of the rewards and both significantly outperform A2C. We note that due to the intrinsic stability of MICARL, it shows even less oscillating behaviour. The loss curves of MICARL and DQN are very similar for Acrobot-v1, while, we note that for Pendulum-v1 MICARL is fitting much better than DQN the loss function.

MountainCar-v0 We show the performance of MICARL on MountainCar-v0 with $n = 50$ and $\eta = 0.1$, but the results for all the other hyperparameters are very similar. Due to its very sparse reward, MountainCar-v0 is usually a very hard task for reinforcement learning agents. In this case, the normalized entropy of the policy is very close to one. This indicates that the policy is sampling actions from the uniform distribution, which seemingly prevents the discovery of any postive reward. As all rewards that the agent observes are -1, the learned Q-function is a completely flat function, which according to Eq. (2) results in a uniform distribution, explaining why the entropy stays close to one. All in all, while entropy regularization is a good heuristic for maintaining high entropy and sustaining exploration, we can see that it does not fully address the exploration problem in RL.

Summary of results. To summarize, we can categorize the results in three. In i) the best case scenario, on CartPole-v1, we observed formidable convergence on all metrics: the cumulative reward reaching and staying at its maximal value, the mean squared Bellman residual going to 10^{-3}, and the KL-divergence between successive policies reaching zero. In ii) the middle case on Pendulum-v1 and Acrobot-v1, while results are on par with the state-of-the-art with perhaps an ever so slightly higher stability of the cumulative rewards, there still remains a persistent Bellman error despite the ever growing size of the Cascade-NN and an abundance of data on relatively simple problems. This is somehow unsatisfying and indicates that research is still needed to discover better policy evaluation algorithms. Finally, in iii) the failure case on MountainCar-v0, where the policies at all iterations are close to uniform over the action space, suggesting that while entropy regularization is a good exploration mechanism with theoretical convergence guarantees, in practice, other exploration methods might still be necessary in some more challenging sparse reward cases.

5. Related Work

Examples of deep RL algorithms implementing entropy regularization include TRPO (Schulman et al., 2015), SAC (Haarnoja et al., 2018) and MPO (Abdolmaleki et al., 2018), but the closest to our work is POLITEX (Abbasi-Yadkori et al., 2019; Lazic et al., 2021). Similarly to MICARL, in POLITEX, the new policy is defined as a Boltzmann distribution over the sum of all past state-action value estimates, resulting from a KL-divergence regularization on the policy update, which makes the learning process less noisy. For example, the experimental results of (Lazic et al.,

2021) show good convergence results, outperforming other policy optimization algorithms. However, to perform this policy update in closed form, Abbasi-Yadkori et al. (2019) considered learning separate state-action value networks which is a computationally expensive process requiring to store all past NNs corresponding to different Q-functions. The more recent implementation of (Lazic et al., 2021) avoids keeping different NNs and instead relies on one NN and experience replay buffer to approximate directly the average behaviour of all previous state-action value functions.

The strategy of POLITEX requires the knowledge of all previously trained state-action value functions. Implementing them with NNs is challenging as the candidate NN should be able to approximate equally good all the old functions together with the new one. This particular problem known as catastrophic forgetting is studied in the field of Incremental Learning. Among the approaches of Incremental Learning, one could distinguish three main directions: architectural, regularization and rehearsal strategies. Architectural strategies, as used in MICARL, suggest modifying (e.g. expanding) the NN structure in order to preserve the good performance on the old tasks and achieve good precision on the new one (Rusu et al., 2016; Fahlman and Lebiere, 1989). Regularization techniques may be divided in two groups. The first group is weight regularization, which is typically done by introducing the additional component in the loss whose goal is to penalize the change of the weights that are important for the old tasks (e.g. (Kirkpatrick et al., 2017; Aljundi et al., 2018)). Second group is knowledge distillation (Li and Hoiem, 2017; Lee et al., 2019), which was firstly used for transfer learning, but can be efficiently applied to Incremental Learning, by bringing the "knowledge" from the old model to the new one, forcing the output of the new model to be more consistent with the output of old models. Finally, rehearsal methods (Rebuffi et al., 2017) or pseudo-rehearsal methods (Mellado et al., 2019) alleviate the problem of catastrophic forgetting by reusing the data of the old tasks when learning the new one. Pseudo-rehearsal methods slightly differ from rehearsal methods as they generate data from the old tasks instead of storing it. More thorough overview of Incremental Learning methods can be found in (Luo et al., 2020).

6. Conclusion

In this paper, we consider entropy regularized reinforcement learning algorithms and how to implement them efficiently in practice. These algorithms were shown to be more stable than the classical approaches as they constrain the current policy to be closer to the previous ones. In our study, we concentrate on the POLITEX algorithm that builds the policy from the output of all Q-value functions of past policies. Implementing those functions in practice is not trivial, and straightforward approaches of assigning one neural network per Q-value function is only possible for very small tasks and cannot scale. Instead, we suggest using MICARL based on Cascade-NN, a neural architecture that grows each iteration by n neurons, capable of preserving past information while new neurons can leverage increasingly richer feature representations. Our preliminary results show that this novel approach can successfully compete with the state-of-the-art for most of the considered use cases, while exhibiting impressive convergence on all considered metrics for CartPole-v1. We believe these are very promising preliminary results suggesting that MICARL and its non-parametric approach to Q-function approximation is worth further investigation. In future work, we would like to understand what makes the algorithm better approximate the Q-function on CartPole-v1 than the other problems, and research the integration of more sophisticated exploration mechanisms in MICARL to tackle sparse reward problems.

Acknowledgments

The authors would like to acknowledge the financial support of the French Ministry of Higher Education and Research, Inria, the Hauts-de-France region. Philippe Preux is also supported by the Métropole Européenne de Lille, through the AI chair Apprenf number R-PILOTE-19-004-APPRENF. Riccardo Della Vecchia is thankful for the funding received by the CHIST-ERA Project Causal eXplainations in Reinforcement Learning – CausalXRL. [3] Alena Shilova acknowledges the founding coming by the Challenge HPC-BigData INRIA Project LAB. [4] We also thank the Scool team at Inria Lille Nord Europe.

3. https://www.chistera.eu/projects/causalxrl
4. https://project.inria.fr/hpcbigdata/

References

Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702. PMLR, 2019.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations (ICLR)*, 2018.

R. Akrour, A. Abdolmaleki, H. Abdulsamad, J. Peters, and G. Neumann. Model-free trajectory-based policy optimization with monotonic improvement. *Journal of Machine Learning Resource (JMLR)*, 2018.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

J. A. Bagnell and J. C. Schneider. Covariant Policy Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Shalabh Bhatnagar, Richard S. Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 2009.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

Omar Darwiche Domingues, Yannis Flet-Berliac, Edouard Leurent, Pierre Ménard, Xuedong Shang, and Michal Valko. rlberry - A Reinforcement Learning Library for Research and Education, 10 2021. URL https://github.com/rlberry-py/rlberry.

Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in neural information processing systems*, 2, 1989.

Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized Markov decision processes. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Sertan Girgin and Philippe Preux. Basis function construction in reinforcement learning using cascade-correlation learning architecture. In *2008 Seventh International Conference on Machine Learning and Applications*, pages 75–82. IEEE, 2008.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

Nevena Lazic, Dong Yin, Yasin Abbasi-Yadkori, and Csaba Szepesvari. Improved regret bound and experience replay in regularized policy iteration. In *International Conference on Machine Learning*, pages 6032–6042. PMLR, 2021.

Kibok Lee, Kimin Lee, Jinwoo Shin, and Honglak Lee. Overcoming catastrophic forgetting with unlabeled data in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 312–321, 2019.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2014.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

Yong Luo, Liancheng Yin, Wenchao Bai, and Keming Mao. An appraisal of incremental learning methods. *Entropy*, 22(11):1190, 2020.

Diego Mellado, Carolina Saavedra, Steren Chabert, Romina Torres, and Rodrigo Salas. Self-improving generative artificial neural network for pseudorehearsal incremental class learning. *Algorithms*, 12(10):206, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *CoRR*, 2017.

J. Peters and S. Schaal. Natural Actor-Critic. *Neurocomputation*, 2008.

Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *International Conference on Machine Learning (ICML)*, 2013.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*.

John Schulman, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *International Conference on Machine Learning (ICML)*, 2015.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010.

Voot Tangkaratt, Abbas Abdolmaleki, and Masashi Sugiyama. Guide actor-critic for continuous control. In *International Conference on Learning Representations (ICLR)*, 2018.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *International Conference on Robotics and Automation (ICRA)*, 2014.

E. Todorov and Weiwei L. A generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems. In *American Control Conference (ACC)*, 2005.

Appendix A. Number of added neurons

We ran an additional set of experiments to study the performance of MICARL with respect to its dependence on n, the number of neurons added per iteration, in Figure 5. In general, we expect better performance as we increase the number of neurons since it should be easier in this case to approximate the Q function. This is confirmed by our simulations for $n \in \{10, 20, 50\}$, where we see that more neurons seem to indicate faster learning, as least initially, while in some cases, the mean squared Bellman residual seem to increase with time for higher n, perhaps due to an overfitting problem. Asymptotically however, the curves of the rewards achieve approximately the same values.

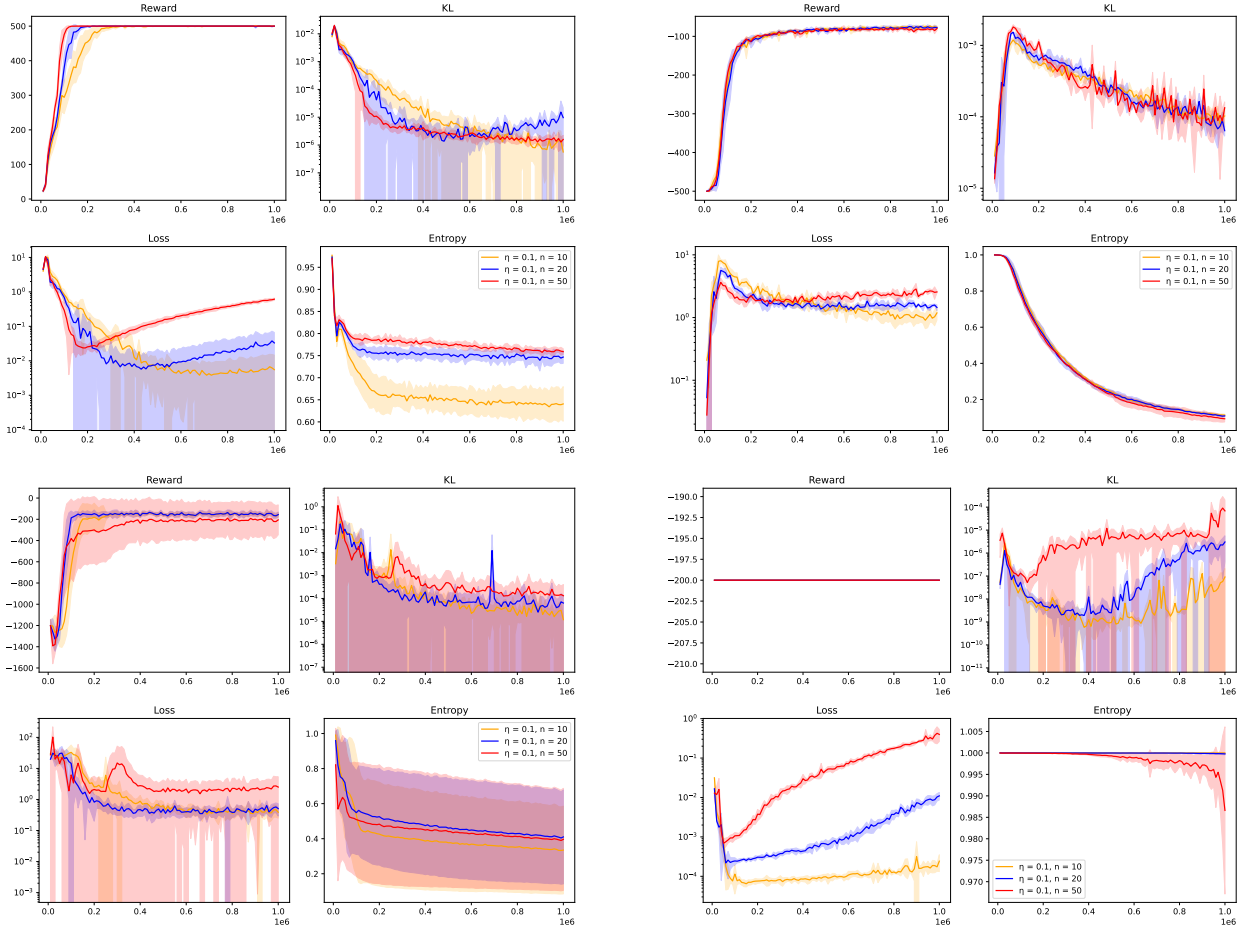


Figure 5: Dependence on the number of added neurons per iteration n on CartPole-v1 (top-left corner), Acrobot-v1 (top-right corner), Pendulum-v1 (bottom-left corner) and MountainCar-v0 (bottom-right corner).