Solving the scalarization issues of Advantage-based Reinforcement Learning Algorithms

Federico A. Galatolo*, Mario G.C.A. Cimino, Gigliola Vaglini

Department of Information Engineering, University of Pisa, 56122 Pisa, Italy

Abstract

In this research, some of the issues that arise from the scalarization of the multi-objective optimization problem in the Advantage Actor Critic (A2C) reinforcement learning algorithm are investigated. The paper shows how a naive scalarization can lead to gradients overlapping. Furthermore, the possibility that the entropy regularization term can be a source of uncontrolled noise is discussed. With respect to the above issues, a technique to avoid gradient overlapping is proposed, while keeping the same loss formulation. Moreover, a method to avoid the uncontrolled noise, by sampling the actions from distributions with a desired minimum entropy, is investigated. Pilot experiments have been carried out to show how the proposed method speeds up the training. The proposed approach can be applied to any Advantage-based Reinforcement Learning algorithm.

Keywords: Reinforcement Learning, Actor Critic, Deep Learning, Gradient-based optimization

1. Introduction and formal background

1.1. Introduction

In last years, unprecedented results has been achieved in the Reinforcement Learning (RL) research field with the use of Artificial Neural Networks (ANNs). In essence, in an RL model an agent interacts with its environment and, upon observation of the consequences of its actions, learns to adapt its own behaviour to rewards received. An agent behavior is modelled in terms of state-action relationships. The goal of the agent is to learn a control strategy (i.e., a policy) maximizing the total reward. An important advancement in the field has been the possibility to operate with high-dimensional state and action spaces via Deep Learning [1].

*Corresponding author

Email addresses: federico.galatolo@ing.unipi.it (Federico A. Galatolo), mario.cimino@unipi.it (Mario G.C.A. Cimino), gigliola.vaglini@unipi.it (Gigliola Vaglini)

More specifically, *policy gradient* models optimize the policy, represented as a parameterized function, via gradient-descent optimization. An increasing interest of the research community has recently led to the paradigm shift of multi-objective reinforcement learning (MORL), in which learning control policies are simultaneously optimized over several criteria [2] [3].

In RL *Advantage learning* is used to estimate the advantage of performing a certain action. [4] Consequently, in the *Actor-Critic (AC)* method a value function(which measures the expected reward) is learned in addition to the policy, in order to assist the policy update [5]. This model is based on a "Critic", which estimates the value function, and an "Actor", which updates the policy distribution in the direction suggested by the "Critic" [6].

This research work focuses on some significant issues of the Advantage Actor Critic (A2C) algorithm, that arise from the scalarization of the multi-objective optimization problem. Firstly, it shows that a naive scalarization can lead to gradients overlapping. Secondly, it investigates the possibility that the entropy regularization term can inject uncontrolled noise. With respect to such issues, a technique to avoid gradient overlapping (called Non-Overlapping Gradient, NOG) is proposed, which keeps the same loss formulation. Moreover, a method to avoid the uncontrolled noise, by sampling the actions from distributions with a desired minimum entropy (called Target Entropy, TE), is investigated. Experimental results compare the A2C algorithm with the proposed combination of A2C with NOG and TE ($\text{A2C}_{\text{NOG+TE}}$).

With regard to performance evaluation, we carried out the hyperparameters optimization for each scenario over the same task [7]. Then using the best hyperparameters, we computed the confidence intervals over multiple runs.

As a relevant result, the combination of TE and NOG determines a decrease of the training time necessary to solve the problem. Specifically, the proposed technique achieves a larger speedup for increasing problem complexity.

The algorithmic design of the proposed approach is compliant with any Advantage-based Reinforcement Learning algorithm derived from A2C that share the same loss function components. The $\text{A2C}_{\text{NOG+TE}}$ algorithm has been developed, tested and publicly released on the Github platform, to foster its application on various research environments.

1.2. Formal background

An RL problem defines an *environment* representing a task. The objective of an RL algorithm is to find an optimal *policy* that an agent has to follow to solve the task. The *environment* can be represented as a Markov Decision Process (MDP). Denoting by S the state space, and by A the action space, it can be defined: (i) the *state transition function* $f_s(s, a) : \mathcal{S} \times \mathcal{A} \Rightarrow \mathcal{S}$; (ii) the reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \Rightarrow \mathbb{R}$.

The objective of an RL algorithm is then to find a policy $\pi(s) : \mathcal{S} \Rightarrow \mathcal{A}$ such that following its trajectories $\mathcal{T} = \{a_t = \pi(s_t), s_{t+1} = f_s(s_t, a_t) \quad \forall t\}$ the cumulative sum of the rewards $\sum_{k=0}^{\infty} r(s_k, a_k)$ for any starting state s_0 is maximized.

Usually, the policy is stochastic: $\pi(s)$ is a function that, for each state $s \in \mathcal{S}$, returns the probability of each action $a \in \mathcal{A}$, i.e., $\pi(s) : \mathcal{S} \Rightarrow \mathcal{A} \times (0, 1)$. By using $\pi(s, a)$ we assume that a is the action *sampled* from a categorical distribution with probabilities $\pi(s)$, and $\pi(s, a) : \mathcal{S} \times \mathcal{A} \Rightarrow (0, 1)$ is the probability of the action a in the distribution $\pi(s)$. Under such assumption, the objective is to maximize the *expectation* of the cumulative sum of the rewards, i.e., $\mathbb{E}[\sum_{k=0}^{\infty} r(s_k, a_k)] = \sum_{k=0}^{\infty} r(s_k, a_k)\pi(s_k, a_k)$.

In the literature, if the *policy* $\pi(s)$ is approximated using an ANN, the term Deep Reinforcement Learning is used. RL algorithms are divided into two major categories: *off-policy* and *on-policy* [8]. The *off-policy* algorithms use stochastic techniques, for example $\epsilon - greedy$, to explore the state space. In the first phase, such algorithms perform random actions and accumulate the *transactions* in a *replay memory*. In the second phase, the *off-policy* algorithms sample some *transactions* from the *replay memory*, and use them to train the *policy*. In contrast, the *on-policy* algorithms explore the space by following the *policy* and updating it via the current *transactions* without a *replay memory*.

In this paper we focus on the issues that arise in a family of *on-policy* algorithms.

1.3. The Advantage Actor Critic (A2C) Algorithm

The *Advantage Actor Critic* (A2C) algorithm, proposed by *OpenAI*, is the synchronous version of the *Asynchronous Advantage Actor Critic* (A3C) algorithm, proposed by *Google* [6]. It has been shown that A2C has the same performance of A3C but with a lower implementation and execution complexity.

A2C is based on the REINFORCE algorithm [5]. Let us define, for each time step t, the *future discounted cumulative reward* $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$. In the REINFORCE algorithm, each optimization step tends to maximize the expectation $E[R_t]$. Let us denote θ_π the parameters of $\pi(s)$. The REINFORCE algorithm follows the optimization trajectory defined by $\Delta_{\theta_\pi} log(\pi(s, a|\theta_\pi))R_t$, which is an unbiased estimation of $\Delta_{\theta_\pi} E[R_t]$ [1].

Usually, the quantity $log(\pi(s, a|\theta_\pi))R_t$ has an high variance, and the optimization trajectories defined by $\Delta_{\theta_\pi} log(\pi(s, a|\theta_\pi))R_t$ are very noisy. To overcome this issue a *baseline* $b(t)$ is used to reduce the variance, and the gradient $\Delta_{\theta_\pi} log(\pi(s, a|\theta_\pi))(R_t - b(t))$ is computed. A classical *baseline* can be the mean of R_t.

The contributions of A2C to REINFORCE are twofold: to use an ANN $V(s_t)$ approximating R_t as the *baseline* $b(t)$, and to use this ANN to bootstrap the R_t computation in partially observed environmental trajectories.

In REINFORCE R_t can be computed after the end of the episode. In contrast, in A2C the $V(s_t)$ estimates R_t, and this value can be used to estimate the *future discounted cumulative reward* before the end of the episode. Therefore, A2C performs an optimization step every N steps, without waiting for the

[1] This is known as the *log derivative trick*.

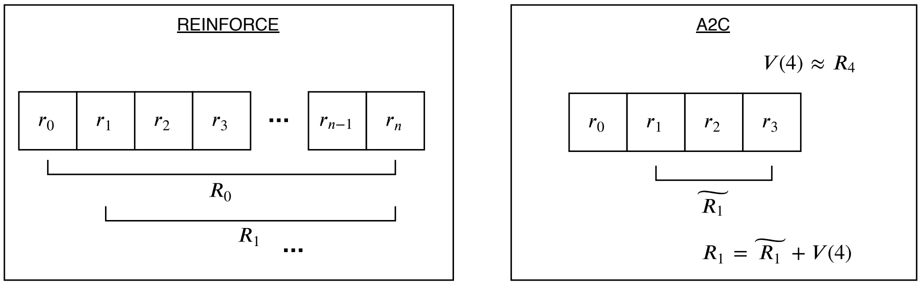


Figure 1: R_t computation in REINFORCE (left) and A2C[2]
(right)

end of the episode. A visual representation of this difference is given in Figure 1. Here, each box represents the current reward r_t whereas R_t represents the future total discounted cumulative reward. In the case of A2C, the future total discounted cumulative reward is computed via the available cumulative reward $\tilde{R}_t$ and an estimation of R_t of the last available state using V.

Overall, the remainder of this paper is structured as follows. Section 2 is devoted to the scalarization issues of the A2C algorithm. The proposed A2C$_{\text{NOG+TE}}$ algorithm is presented in Section 3. Experimental studies are covered by Section 4. Finally, Section 5 summarizes the major achievements and future work.

2. Scalarization issues of the A2C algorithm

The A2C algorithm uses two ANNs to approximate the two functions $\pi(s|\theta_\pi)$ and $V(s|\theta_v)$. As previously stated, in A2C the environment is observed only for N steps (instead of waiting for the episode termination). Given the partial *state-action-reward* $(s_k, a_k, r_k) \forall k \in t_s, \dots, t_s + N$ observation, the algorithm computes, for each k:

1. R_k using $V(s_{N+1})$ as bootstrap: $R_k = \sum_{i=k}^{N} \gamma^{i-k} r_k + \gamma^{N-k} V(s_{N+1})$;
2. The policy gradient $\Delta_{pg} = \Delta_{\theta_\pi} log(\pi(s_k, a_k|\theta_\pi))(R_k - V(s_k))$;
3. The $V(s|\theta_v)$ gradient $\Delta_v = \Delta_{\theta_v}(V(s_k|\theta_v) - R_k)^2$;
4. The entropy gradient $\Delta_h = \Delta_{\theta_\pi} \sum_{i=0}^{N} log(\pi(s_i, a_i|\theta_\pi))\pi(s_i, a_i|\theta_\pi)$.

Subsequently, an optimization step is performed in the direction that maximizes both $\mathbb{E}[R_k]$ (direction Δ_{pg}) and the entropy of $\pi(s_k)$ (direction Δ_h), as well as minimizes the mean squared error of $V(s_k)$ (direction $-\Delta_v$). It is a multi-objective optimization problem, which in the A2C algorithm has been solved with a *scalarization.* There are three different objectives, with some common parameters. Both the entropy and policy gradients share θ_π.

[2] For simplicity $R_t = \sum_{i=0}^{\infty} r_{t+i}$ is used in this example

Also $\pi(s)$ and $V(s)$ often have some common parameters, because usually a feature extraction is performed on the state s, and the features are used as inputs for $\pi(s)$ and $V(s)$. Let us denote $C(s|\theta_C) : \mathcal{S} \Rightarrow \mathcal{F}$ the feature extraction function, with θ_C its parameters, $f = C(s|\theta_C)$ the features. By substituting $\mathcal{S}$ with in $\mathcal{F}$ in the $\pi(s)$ and $V(s)$ domains [3], then the computed gradients are:

$$\Delta_{pg} = \Delta_{\theta_\pi + \theta_C} log(\pi(f_k, a_k|\theta_\pi, \theta_C))(R_k - V(f_k)) \tag{1}$$

$$\Delta_v = \Delta_{\theta_v + \theta_C}(V(f_k|\theta_v, \theta_C) - R_k)^2 \tag{2}$$

$$\Delta_h = \Delta_{\theta_\pi + \theta_C} \sum_{i=0}^{N} log(\pi(s_i, a_i|\theta_\pi, \theta_C))\pi(s_i, a_i|\theta_\pi, \theta_C) \tag{3}$$

where Δ_{pg} is the policy gradient, Δ_v is the error gradient for the estimator net V, and Δ_h is a gradient of the entropy of the policy net. The notation $\Delta_{\theta_\pi + \theta_C}(\cdot)$ represents the gradient of the argument with respect to θ_π and θ_C.

An optimization step is performed in the direction of the scalarized objective $-\Delta_{pg} + \beta\Delta_v - \alpha\Delta_h$, where α and β are coefficients introduced to weight the strength of the entropy regularization term and of the Δ_v gradient, respectively. It is apparent that all the three objective functions share some parameters. Specifically, the gradient computed for the parameter θ_π contains the contributions of Δ_{pg} and Δ_h. Furthermore, the gradient for the parameter θ_C contains the contributions of Δ_{pg}, Δ_v and Δ_h.

A representation of the mutual dependency between gradients via related parameters is given in Figure 2.

Each coloured box represents a different gradient contribution to the overall loss related to: the policy π, the entropy h, and the total discounted cumulative reward estimator V. Here, each big box represents a different Neural Network (NN), whereas the inner small box represents its parameters (i.e. a connection weights). In Figure, the input and output of each NN are also represented: $C(s)$ is fed by the state s to extract the features f, whereas both the policy NN π and the estimator NN V take the features as an input, to provide the action probability vector p and the future cumulative discounted reward estimate $\tilde{R}$, respectively. In particular, each gradient is represented with a different color, and a dashed colored arrow from the gradient to the inputs highlights the backward path and thus the influence of a gradient to a parameter optimization. It is apparent that the sub-objectives are not independent, since they have common parameters. We call gradient overlapping this dependency among gradients. As a consequence, the policy and value function parameters can be pushed to sub-optimal regions.

[3] $\pi(f) : \mathcal{F} \Rightarrow \mathcal{A} \times (0, 1)$, $\pi(f, a) : \mathcal{F} \times \mathcal{A} \Rightarrow (0, 1)$ and $V(f) : \mathcal{F} \Rightarrow R$

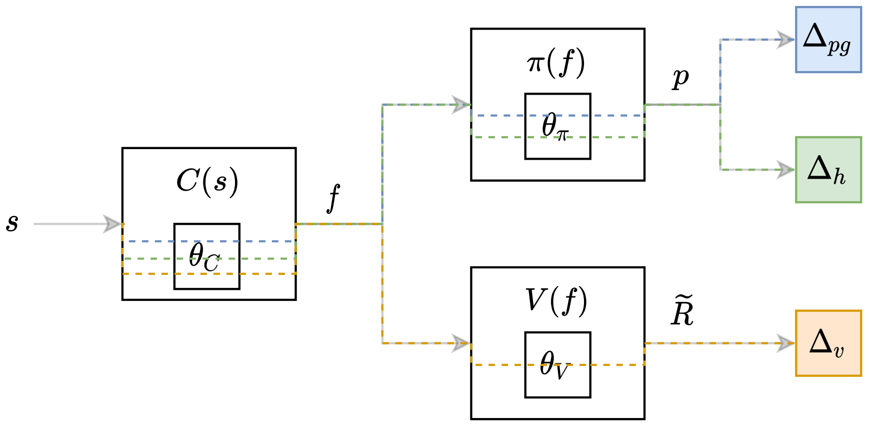


Figure 2: Backward computation in the A2C algorithm

Another issue that is considered in this research is the possibility that the entropy regularization term could generate noise in the network parameters. Indeed, it can be observed in Formula 3 that the gradient Δ_h is not computed to reach a target entropy level, but just for increasing it.

In the next section the two issues are tackled considering also their reciprocal impact on the system performance.

3. The Proposed A2C$_{\mathbf{NOG+TE}}$ algorithm

In this section a solution to avoid the gradient overlapping when using the A2C scalarized objective function is proposed. It is worth noting that to solve the gradient overlapping problem also allows to remove the weights coefficients of the scalarized objective function, thus reducing the hyperparameters search space, and then the optimization time. In the following, this approach will be referred to as the Non-Overlapping Gradient (NOG). Furthermore, an idea to solve the noise generated by the entropy regularization term is discussed. The idea is to maintain the entropy of the policy $\pi(f)$ above a target level without using any gradient. In the following, this approach will be referred to as the Target Entropy (TE). As an effect, this can further reduce the gradient overlapping phenomenon.

3.1. Non-Overlapping-Gradients (NOG)

The NOG technique consists in simplifying the backward computation flow represented in Fig. 2, to remove the gradient overlapping on the feature extraction function $C(s)$, and to constrain the computation to the semantically appropriate functions. Specifically, the only gradient contributing to the feature extraction function $C(f)$ optimization is Δ_{pg}. Similarly, the gradient Δ_h should contribute just to the policy function $\pi(f)$ optimization, as well as the gradient Δ_v should contribute just to the value function $V(f)$ optimization. According to such criterion, the new computed gradients are the following:

$$\Delta_{pg} = \Delta_{\theta_\pi + \theta_C} log(\pi(f_k, a_k|\theta_\pi, \theta_C))(R_k - V(f_k)) \tag{4}$$

$$\Delta_v = \Delta_{\theta_v}(V(f_k|\theta_v) - R_k)^2 \tag{5}$$

$$\Delta_h = \Delta_{\theta_\pi} \sum_{i=0}^{N} log(\pi(s_i, a_i|\theta_\pi))\pi(s_i, a_i|\theta_\pi) \tag{6}$$

where, with respect to Formulas 1, 2, 3, Δ_v and Δ_h are computed respectively against θ_v and θ_π only.

This way, the gradient overlapping is sensibly reduced, but not totally disappeared. Specifically in this scenario the gradients Δ_{pg} and Δ_h still overlap via the parameters of the policy function θ_π. A visual representation of the new gradient computation is given in Figure 3, where a colored cross represents where the backward computation of the related gradient component stops.

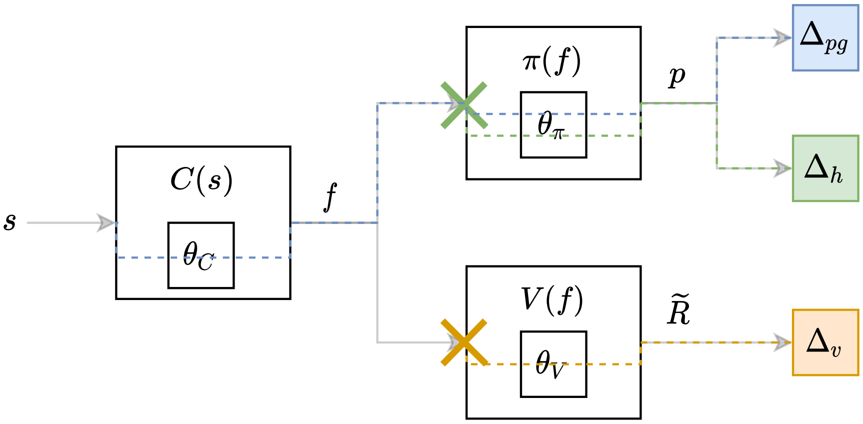


Figure 3: Backward computation in the A2C$_{\text{NOG}}$ algorithm

Using the *Non Overlapping Gradients* technique the new scalarized objective function is $-\Delta_{pg} + \Delta_v - \alpha\Delta_h$. Note that the parameter β is not needed because Δ_v is totally independent.

3.2. Target Entropy (TE)

In the previous section, it has been highlighted that the gradient Δ_h is not computed to reach a target entropy level but just for increasing it. This can produce noise in the network parameters. In this section we propose a novel technique to maintain the entropy of the policy $\pi(f)$ above a target level without using any gradient. As a consequence, the gradient overlapping can be completely removed when using the TE technique in conjunction with *NOG*.

Let us denote $p_a = \pi(f, a)$ the probabilities of each action $a \in \mathcal{A}$ given the features $f = C(s)$ of the state $s \in \mathcal{S}$, and p_{max} the highest probability. Let us observe that $\sum_i^N p_i = 1$. Let us define $\tilde{p}$ as:

$$\tilde{p}_i = \begin{cases} p_i - \epsilon & i = max \\ p_i + \frac{\epsilon}{N-1} & i \neq max \end{cases} \tag{7}$$

The property $\sum_i^N \tilde{p}_i = 1$ is maintained [4], i.e., $\tilde{p}$ is still a valid categorical distribution. It is also important to notice that $H(\tilde{p}) < H(p)$.

Let us recall the definition of entropy $H(x) = -\sum_i^N log(x_i)x_i$, and let us focus on just one of the entropy components $log(x)x$. It can be easily compute the difference of one contribution in function of ϵ $\Delta h(x, \epsilon) = log(x)x - log(x + \epsilon)(x + \epsilon)$. Considering the overall entropy difference $\Delta H(p, \epsilon) = H(p) - H(\tilde{p}|\epsilon)$, it can be written in function of $\Delta h(p, \epsilon)$ contributions, as follows:

$$\begin{aligned}
\Delta H(p, \epsilon) =& \sum_i^N log(p_i)p_i - \sum_i^N log(\tilde{p_i})\tilde{p_i} \\
\Delta H(p, \epsilon) =& log(p_0)p_0 + \dots + log(p_n)p_n + \\
& - (log(p_0 + \frac{\epsilon}{N-1})(p_0 + \frac{\epsilon}{N-1}) + \cdots \\
& + log(p_{n-1} + \frac{\epsilon}{N-1})(p_{n-1} + \frac{\epsilon}{N-1}) + log(p_{max} - \epsilon)(p_{max} - \epsilon))
\end{aligned}$$

Rearranging the terms, $\Delta H(p, \epsilon)$ can be rewritten as:

$$\begin{aligned}
\Delta H(p, \epsilon) =& (log(p_0)p_0 - log(p_0 + \frac{\epsilon}{N-1})(p_0 + \frac{\epsilon}{N-1})) + \\
& \dots \\
& + (log(p_{n-1})p_{n-1} - log(p_{n-1} + \frac{\epsilon}{N-1})(p_{n-1} + \frac{\epsilon}{N-1})) \\
& + log(p_{max} - \epsilon)(p_{max} - \epsilon)
\end{aligned}$$

Expressing it in function of Δ_h:

$$\Delta H(p, \epsilon) = \Delta_h(p_0, \frac{\epsilon}{N-1}) + \dots + \Delta_h(p_{n-1}, \frac{\epsilon}{N-1}) + \Delta_h(p_{max}, -\epsilon)$$

Let us assume that ϵ is small and close to zero. The Taylor expansion of $\Delta_h(p, \epsilon)$ where $\epsilon = 0$ can be computed as follows:

[4] $\sum_i^N \tilde{p}_i = \sum_{i \neq max} \tilde{p}_i + p_{max} = \sum_{i \neq max} p_i - (N-1)\frac{\epsilon}{N-1} + p_{max} - \epsilon = \sum_{i \neq max} p_i + p_{max} - \epsilon + \epsilon = \sum_i^N p_i = 1$

$$\frac{\partial}{\partial\epsilon}\Delta_h(p,\epsilon) = -(\frac{1}{p+\epsilon}(p+\epsilon) + log(p+\epsilon)) = -log(p+\epsilon) - 1$$
$$\Delta_h(p,\epsilon)|_{\epsilon\sim 0} \approx \Delta_h(p,0) + \frac{\partial}{\partial\epsilon}\Delta_h(p,0)\epsilon$$
$$\Delta_h(p,\epsilon)|_{\epsilon\sim 0} \approx log(p)p - log(p)p + (-log(p) - 1)\epsilon \approx -log(p)\epsilon - \epsilon$$
$$\Delta_h(p,\epsilon)|_{\epsilon\sim 0} \approx -\epsilon(log(p) + 1)$$

Finally, by substituting back the approximation of $\Delta_h(p,\epsilon)$ in $\Delta H(p,\epsilon)$, the following approximation can be derived:

$$\Delta H(p,\epsilon) = \Delta_h(p_0, \frac{\epsilon}{N-1}) + \cdots + \Delta_h(p_{n-1}, \frac{\epsilon}{N-1}) + \Delta_h(p_{max}, -\epsilon)$$
$$\Delta H(p,\epsilon) \approx -\frac{\epsilon}{N-1}(log(p_0) + 1) \cdots - \frac{\epsilon}{N-1}(log(p_{n-1}) + 1) + \epsilon(log(p_{max}) + 1)$$
$$\Delta H(p,\epsilon) \approx -\frac{\epsilon}{N-1}(log(p_0) + log(p_1) + \cdots + log(p_{n-1}) - (N-1)(log(p_{max}) + 1) + (N-1))$$
$$\Delta H(p,\epsilon) \approx -\frac{\epsilon}{N-1}(\sum_i^{n-1} p_i - (N-1)log(p_{max}) - (N-1) + (N-1))$$
$$\Delta H(p,\epsilon) \approx -\epsilon(\frac{\sum_i^{n-1} p_i)}{N-1} - log(p_{max}))$$
$$\Delta H(p,\epsilon) \approx -\epsilon(AVG_{i\neq i_{max}}[p_i] - log(p_{max}))$$

Using the above formula, ϵ can be computed as follows, in order to achieve a desired entropy T_h of p:

$$\epsilon = -\frac{H(p) - T_h}{AVG_{i\neq i_{max}}[p_i] - log(p_{max})} \tag{8}$$

As a consequence, the action can be sampled from $\tilde{p}|\epsilon$ instead of p, i.e., to sample the action from a categorical distribution with an entropy higher than T_h. It is worth to notice that the action from the $\tilde{p}|\epsilon$ distribution can still be sampled using the Δ_{pg} gradient computation represented in Figure 4. As a result, the technique allows to keep a certain *exploration over exploitation* ratio, and at the same time it avoids raising entropy.

Using the *Target Entropy* technique, the new scalarized objective function is $-\Delta_{pg} + \beta\Delta_v$. It can be noted that there is no Δ_h term. Figure 4 represents the resulting backward computation. Here, the focus is on the NN π, whose output p is now transformed using the *Target Entropy* according to 7 and 8. The resulting $\tilde{p}$ is used to sample an action a. As a result, there is no more a contribution to the gradient related to Δ_h. Then, the scalarization coefficients are not needed, because there are only two independent contributions to the gradient.

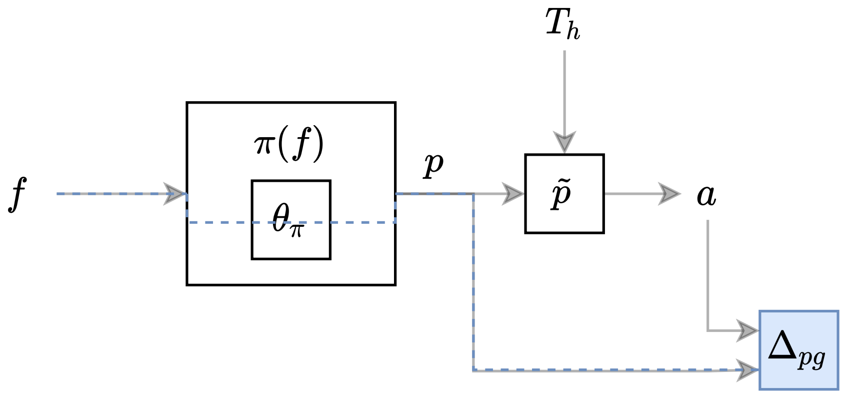


Figure 4: Backward computation in the A2C$_{\text{TE}}$ algorithm

In the next section, the advantages of the NOG and TE techniques are experimentally evaluated.

4. Experimental Studies

In order to investigate the combined effect of the NOG and TE techniques, two different algorithms have been experimented:

1. Classical A2C (A2C)
2. A2C with Non-Overlapping-Gradients and Target Entropy (A2C$_{\text{NOG+TE}}$)

For each training algorithm, first a hyperparameters optimization has been carried out. Subsequently, the best hyperparameters have been used to calculate the confidence interval, over 10 runs, of the training time needed to solve the problem.

To perform the experiments, three environments sufficiently complex to solve, which allow the hyperparameters optimization in a reasonable time, have been considered: *EnergyMountainCar*, *CartPole* and *LunarLander*, all from OpenAI Gym [9].

4.1. Hyperparameters Optimization

Table 1 shows the hyperparameters to optimize, for the considered algorithms. In order to sample the hyperparameters to use for each run, it has been used the Tree-structured Parzen Estimator (TPE) [7], whereas to prune unpromising runs it has been used the Successive Halve Pruning (SHP) [10]. More precisely every 1000 steps the current reward EMA (Exponential Moving Average) is reported to the SHP pruner.

Each run has been evaluated for 100 episodes, and the mean reward has been used as objective function (to maximize) for the hyperparameters optimization. All hyperparameters optimization has been run on an Intel Xeon with 40 cores.

Name	Range	Sampling	Description
γ	$[0.9, 0.99, 0.999]$	Categorical	Discount factor
N	$[8, 16, 32, 64]$	Categorical	Env. steps for training step
lr	$(10^{-5}, 10^{-2})$	LogUniform	Learning rate
mcn	$(0, 2)$	Uniform	Max gradient clip norm
α	$(10^{-4}, 10^{-1})$	LogUniform	Δ_h strength
β	$(0, 1)$	Uniform	Δ_v strength
T_h	$(0, 0.2)$	Uniform	Target Entropy

Table 1: Hyperparameters to optimize.

It follows, for each environment, a brief description, the results of the hyperparameters optimization, and the performance evaluation for the two comparative algorithms.

4.2. The EnergyMountainCar environment

In *EnergyMountainCar* a car drives up a hill which is steep with respect to its engine. Since the car is positioned in a valley, the agent must learn to drive back and forth to build up momentum. Figure 5 shows the environment and its control variables.

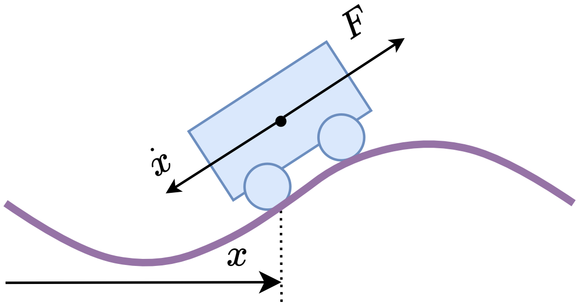
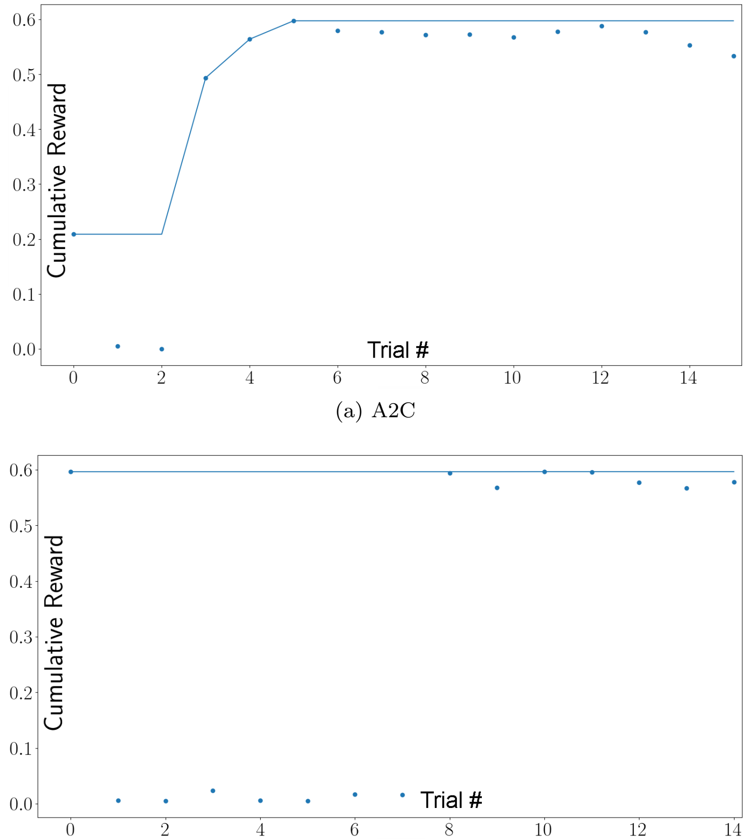


Figure 5: The EnergyMountainCar environment

Specifically, the state space has 2 components: car's horizontal position (x) and horizontal speed ($\dot{x}$). Three different actions can be performed by the agent: no action, accelerate (F) to the left or to the right. The reward is computed as the car's total energy difference (potential and kinetic) of the last time step.

An episode finishes when the car reaches the top of the right hill. The goal is to spend less energy as possible. The environment is considered solved by achieving a cumulative reward of 0.45 points.

Figure 6 shows the objective value of the hyperparameters optimization process, against the number of trials sampled, for the comparative algorithm. The running best objective value is highlighted by a continuous line. In particular, it can be noted that the best objective value is immediately achieved by the $\text{A2C}_{\text{NOG+TE}}$, whereas it is achieved at the fifth iteration by the A2C.

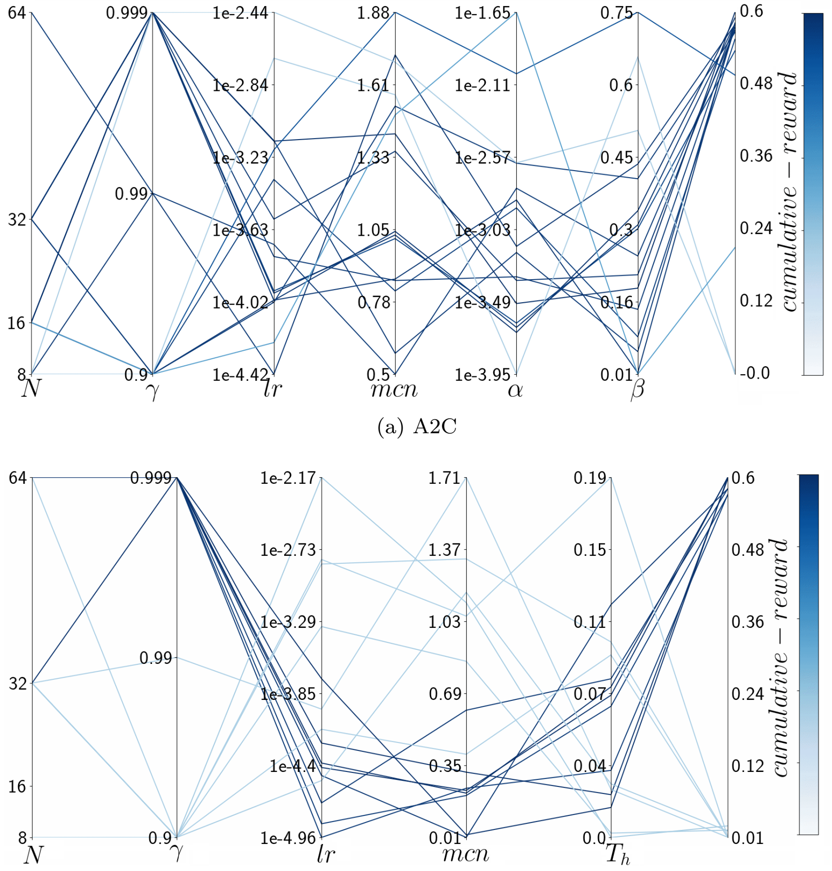


(a) A2C

(b) $\text{A2C}_{\text{NOG+TE}}$

Figure 6: EnergyMountainCar: objective value of the hyperparameters optimization process over time, for the comparative algorithms. The solid line highlights the best values.

Figure 7 represents the hyperparameters values optimization and the related objective value, for the two algorithms. Here, each line represents a trial, with its hyperparameters values represented on the vertical axes. According to the colorbar, the blue level of the line allows to distinguish the best solutions. Here, it can be observed that the hyperparameters values corresponding to the highest objective values are more scattered for the A2C.



(a) A2C

(b) A2C$_{\text{NOG+TE}}$

Figure 7: EnergyMountainCar: hyperparameters values optimization and related objective value, for the comparative algorithms.

For the sake of completeness, Table 2 shows the best hyperparameters value for each considered algorithm.

parameter	A2C	A2C$_{\text{NOG+TE}}$
γ	0.999	0.999
N	16	64
lr	0.0007139	0.00003798
max-clip-norm	1.419	0.2302
α	0.0003160	
β	0.1833	
T_h		0.0739

Table 2: EnergyMountainCar: best hyperparameters found for each algorithm.

After setting the best hyperparameters for each algorithm, the training process has been carried out 10 times for both algorithms.

Figure 8 shows the episode reward versus the training step for each algorithm, with its 95% confidence interval. Precisely, the steps to solve the problem via the proposed $\text{A2C}_{\text{NOG+TE}}$ algorithm and via the classical A2C are 2511 ± 378 and 2702 ± 433, respectively. The proposed approach improves the time efficiency of the A2C, up to more than 1.08x of average speedup.

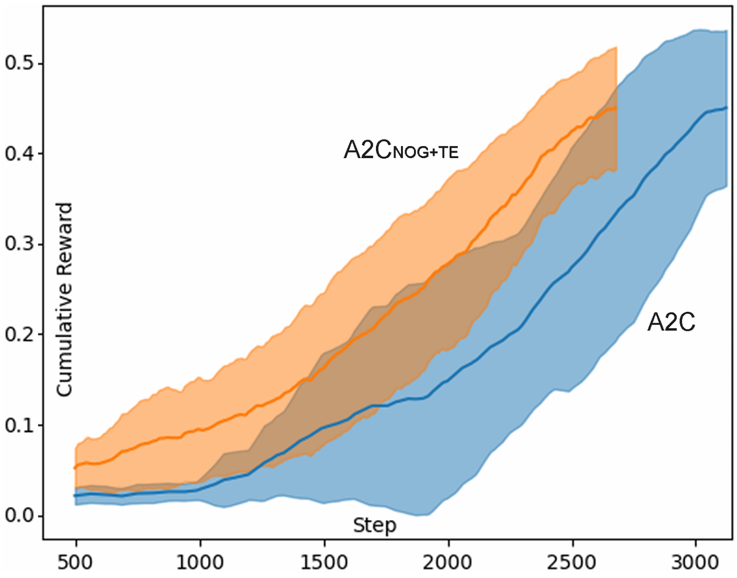


Figure 8: EnergyMountainCar: reward versus training step, for each algorithm.

4.3. The CartPole environment

CartPole, also known as *inverted pendulum*, is a pendulum with the center of mass above its pivot point. Figure 9 shows the environment and its control variables. The pivot point is an axis of rotation mounted on a cart, limiting the pendulum to one degree of freedom, along which the cart can move horizontally. Any displacement from the vertical position causes a gravitation torque and a consequent fall, if not balanced by the cart movement. The agent controls the cart in order to prevent the pendulum from falling, by applying a force F of ± 1. The state space is represented by 4 components: cart position (x), cart velocity ($\dot{x}$), pole angle (Θ), and pole tip angular velocity ($\dot{\Theta}$). The action space is two-dimensional: moving left or right. A reward of +1 is provided for every timestep with the pole upright. An episode ends when the pole is more than 15 degrees from vertical, or when the cart moves more than 2.4 units from the start. The goal is to keep the pole upright as much as possible. The environment is considered solved by achieving a cumulative reward of 195 points.

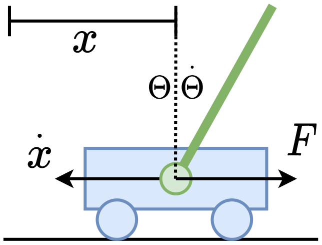
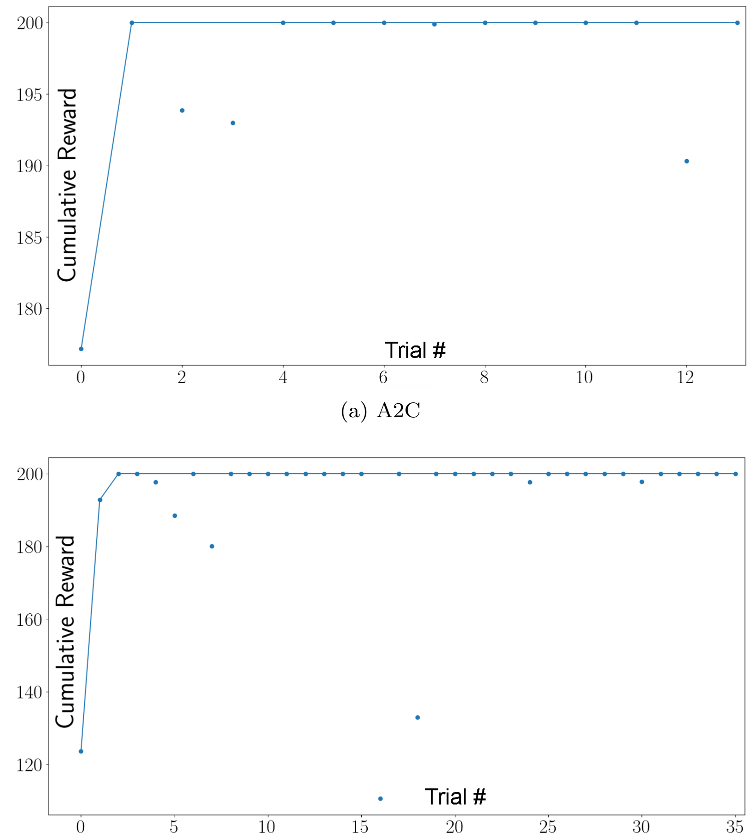


Figure 9: The CartPole environment

Figure 10 shows the objective value of the hyperparameters optimization process, against the number of trials sampled, for the comparative algorithm. It is worth noting that, there is a lower number of trials in the hyperparameters optimization of A2C. Specifically, during the optimization, there are unpromising trials which are aborted during the training process by the pruning algorithm. The figures clearly show that the A2C has been more affected by pruning with respect to the A2C$_{\mathrm{NOG+TE}}$.

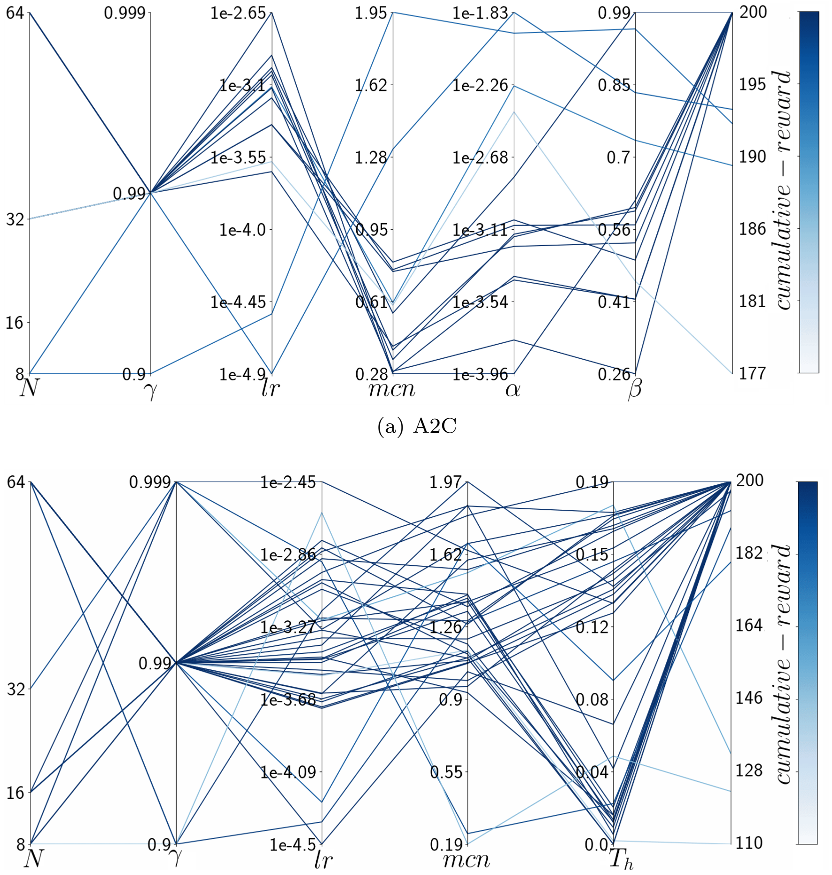


(a) A2C

(b) A2C$_{\mathrm{NOG+TE}}$

Figure 10: CartPole: objective value of the hyperparameters optimization process over time, for the comparative algorithms. The solid line highlights the best values.

Figure 11 represents the hyperparameters values optimization and the related objective value, for the two algorithms. Here, each line represents a trial, with each hyperparameter value represented on the vertical axes. According to the colorbar, the blue level of the line allows to distinguish the best solutions. Here, it can be observed that for some hyperparameters values there is a higher density of good trials.



(a) A2C

(b) A2C$_{\text{NOG+TE}}$

Figure 11: CartPole: hyperparameters values optimization and related objective value, for the comparative algorithms.

For the sake of completeness, Table 3 shows the best hyperparameters value for both algorithms.

parameter	A2C	A2C$_{\text{NOG+TE}}$
γ	0.99	0.99
N	64	64
lr	0.0009591	0.001642
max-clip-norm	0.3898	1.3569
α	0.0006986	
β	0.5996	
T_h		0.166

Table 3: CartPole: best hyperparameters found for each algorithm.

After setting the best hyperparameters for each algorithm, the training process has been carried out 10 times for each algorithm.

Figure 12 shows the episode reward versus the training step for each algorithm, with its 95% confidence interval. Precisely, the steps to solve the problem via the proposed $\mathrm{A2C_{NOG+TE}}$ algorithm and via the classical A2C are 848 ± 197 and 999 ± 108, respectively. The proposed approach sensibly improves the time efficiency of the A2C, up to more than 1.18x of average speedup.

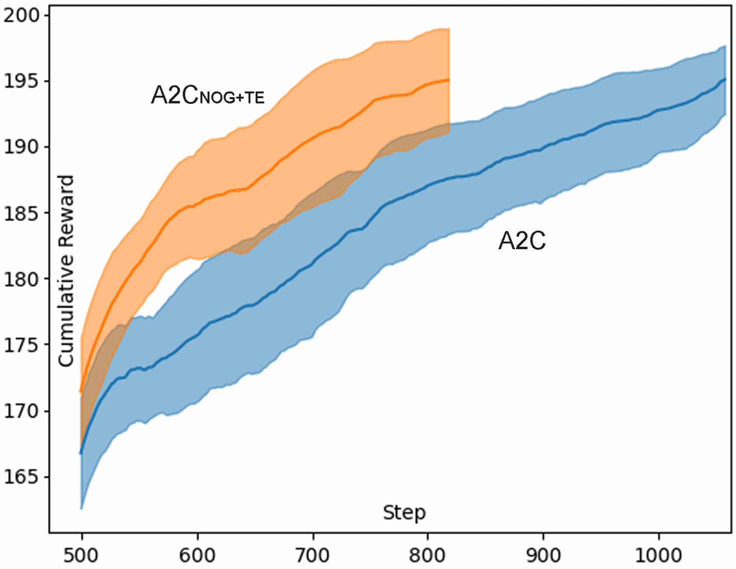


Figure 12: CartPole: reward versus training step, for each algorithm.

4.4. The LunarLander environment

LunarLander is a control task, in which the agent controls the landing of a spacecraft. The spacecraft is initialized at the top of the environment, with a random velocity and angular momentum. Figure 13 shows the environment and its control variables.

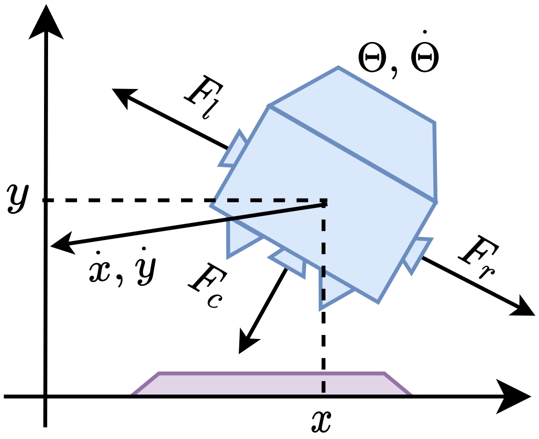
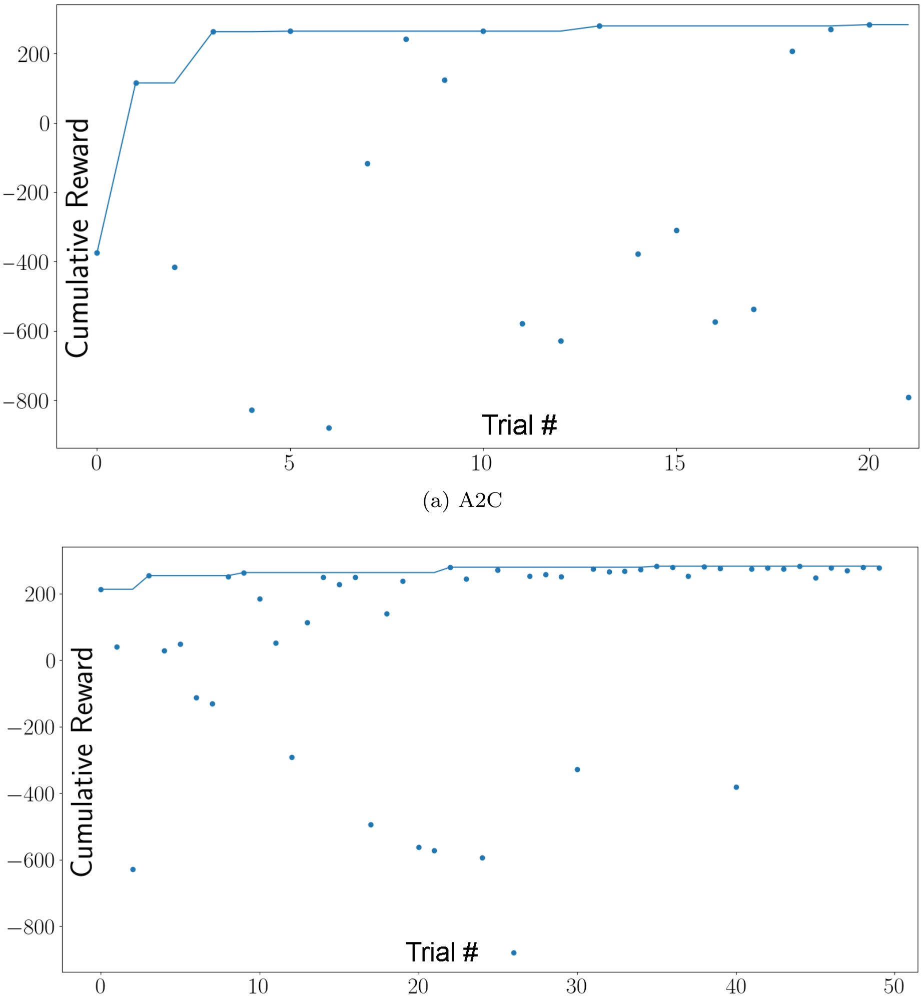


Figure 13: The LunarLander environment

Specifically, the state space has 8 components: horizontal position and velocity $(x, \dot{x})$, vertical position and velocity $(y, \dot{y})$, angle (Θ) and angular momentum $(\dot{\Theta})$, right and left leg state (that is, leg ground contact). Four different actions can be performed by the agent: no action, fire left engine (F_l), fire right engine (F_r) and fire main engine (F_c). The spacecraft has infinite fuel. The reward is computed as follows: -0.3 points for each frame with the main engine on, +100 points for a successful landing, -100 points for crashing, +10 points for each leg making contact with the ground, and a value ranging from 100 to 140 evaluating the spacecraft trajectory to the pad. An episode finishes when the spacecraft lands or crashes. The goal is to land the spacecraft using as less fuel as possible. The environment is considered solved by achieving a cumulative reward of 200 points.

Figure 14 shows the objective value of the hyperparameters optimization process, against the number of trials sampled, for the comparative algorithm. It is worth noting that, actually, good hyperparameters can be found after just 20 trials. It is worth noting that, there is a lower number of trials in the hyperparameters optimization of A2C. Specifically, during the optimization, there are unpromising trials which are aborted during the training process by the pruning algorithm. The figures clearly show that the A2C has been more affected by pruning with respect to the A2C$_{\text{NOG+TE}}$.

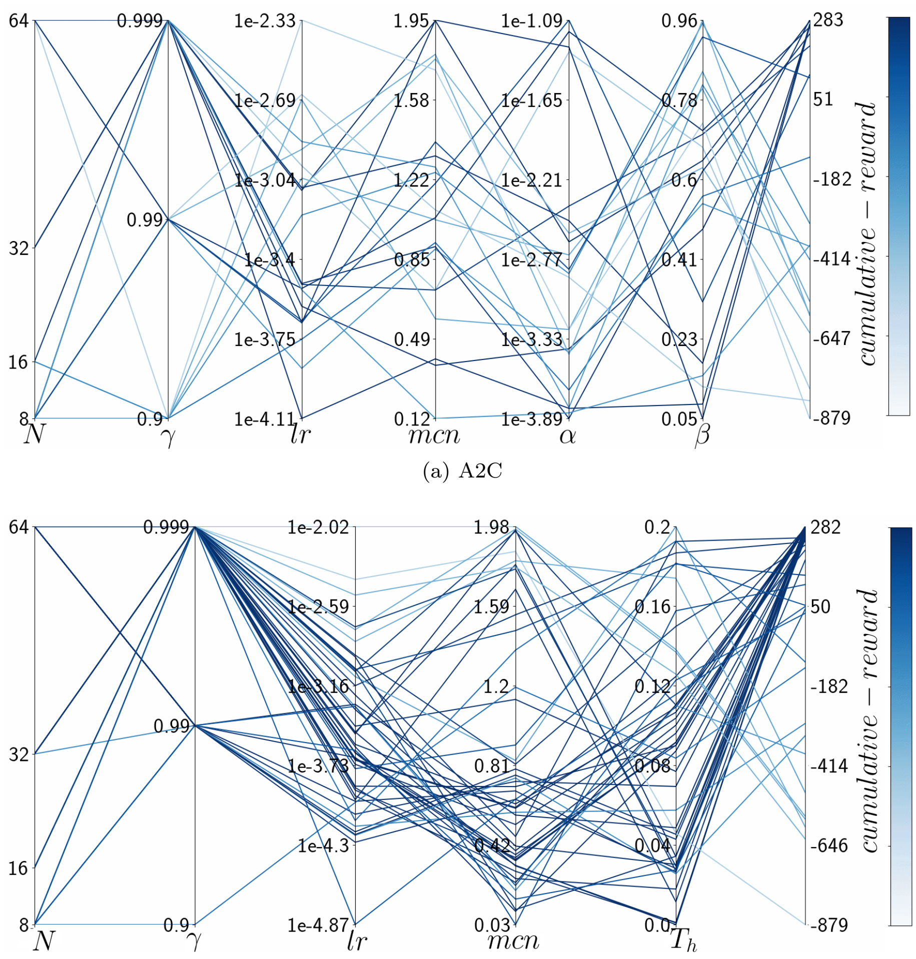


(a) A2C

(b) A2C$_{\mathrm{NOG+TE}}$

Figure 14: LunarLander: objective value of the hyperparameters optimization process over time, for the comparative algorithms. The solid line highlights the best values.

Figure 15 represents the hyperparameters values optimization and the related objective value, for the two algorithms. Here, each line represents a trial, with its hyperparameters values represented on the vertical axes. According to the colorbar, the blue level of the line allows to distinguish the best solutions. In particular, in Figure 15b it can be observed that for some hyperparameters values there is a higher density of good trials.



(a) A2C

(b) $\text{A2C}_{\text{NOG+TE}}$

Figure 15: LunarLander: hyperparameters values optimization and related objective value, for the comparative algorithms.

Table 4 shows the best hyperparameters value for each considered algorithm.

After setting the best hyperparameters for each algorithm, the training process has been carried out 10 times for each algorithm.

Figure 16 shows the episode reward versus the training step for each algorithm, with its 95% confidence interval. Precisely, the steps to solve the problem via the proposed $\text{A2C}_{\text{NOG+TE}}$ algorithm and via the classical A2C are 2045 ± 446 and 6265 ± 2615, respectively. It is apparent that the proposed approach sensibly improves the time efficiency of the A2C, up to more than 3.06x of average speedup.

parameter	A2C	A2C$_{\mathrm{NOG+TE}}$
γ	0.999	0.999
N	64	64
lr	0.0002473	0.0002292
max-clip-norm	0.3668	0.3462
α	0.0003978	
β	0.4832	
T_h		0.0917

Table 4: LunarLander: best hyperparameters found for each algorithm.

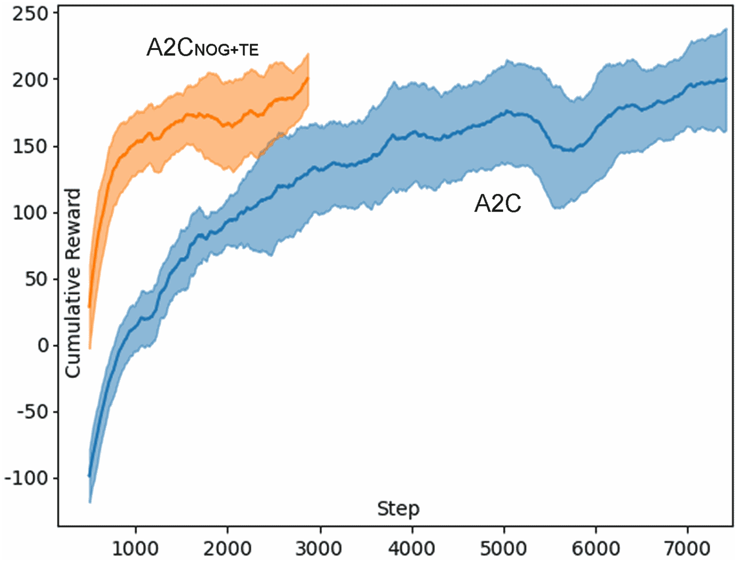


Figure 16: LunarLander: reward versus training step, for both algorithms.

4.5. Results summary

Table 5 summarizes the 95% confidence intervals of the training steps needed to solve the three considered environments, via A2C and A2C$_{\text{NOG+TE}}$, and the speedups with respect to A2C. The effectiveness of the proposed A2C$_{\text{NOG+TE}}$ is apparent for increasing environment complexity (i.e., state space size).

Environment	State space size	A2C	A2C$_{\text{NOG+TE}}$	Average Speedup
EnergyMountainCar	2	2702 ± 433	2511 ± 378	1.08x
CartPole	4	999 ± 108	848 ± 197	1.18x
LunarLander	8	6265 ± 2615	2045 ± 446	3.06x

Table 5: Confidence intervals of the steps to solve some benchmark environments, via A2C and A2C$_{\text{NOG+TE}}$ algorithms.

The A2C$_{\text{NOG+TE}}$ algorithm has been developed, tested and publicly released on the Github platform [11], to foster its application on various research environments.

5. Conclusions

In the Advantage Actor Critic (A2C) algorithm, two issues of the scalarization of the multi-objective optimization problem are discussed and addressed. Specifically, an approach to avoid gradient overlapping (NOG) and to control the entropy (TE) of the action distribution is formally designed. The proposed variant, called A2C$_{\text{NOG+TE}}$, and the classical A2C, are experimented, after performing the hyperparameters optimization.

The proposed techniques are designed to be used on all the reinforcement learning algorithms derived from A2C that share the same loss function components. Although the preliminary experiments look promising, more research is needed to both investigate the performance improvements on different environments and on different Advantage based algorithms.

Acknowledgements

This research was partially carried out in the framework of the following projects: (i) PRA 2018_81 project entitled “Wearable sensor systems: personalized analysis and data security in healthcare” funded by the University of Pisa; (ii) CrossLab project (Departments of Excellence), funded by the Italian Ministry of Education and Research (MIUR); (iii) “KiFoot: Sensorized footwear for gait analysis” project, co-funded by the Tuscany Region (Italy) under the PAR FAS 2007-2013 fund and the FAR fund of the Ministry of Education, University and Research (MIUR).

References

[1] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[2] T. T. Nguyen, "A multi-objective deep reinforcement learning framework," *arXiv preprint arXiv:1803.02965*, 2018.

[3] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems*, pp. 14610–14621, 2019.

[4] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, pp. 5279–5288, 2017.

[5] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[7] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, pp. 2546–2554, 2011.

[8] M. Tokic and G. Palm, "Gradient algorithms for exploration/exploitation trade-offs: Global and local variants," in *Artificial Neural Networks in Pattern Recognition* (N. Mana, F. Schwenker, and E. Trentin, eds.), (Berlin, Heidelberg), pp. 60–71, Springer Berlin Heidelberg, 2012.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[10] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," *CoRR*, vol. abs/1810.05934, 2018.

[11] F. A. Galatolo, "`https://github.com/galatolofederico/a2c-te-nog`," 2020.

Authors

Federico A. Galatolo is a PhD student in Information Engineering at the Department of Information Engineering of the University of Pisa (Italy). His research is focused on Computational Stigmergy, Deep Learning and Reinforcement Learning.

Mario G.C.A. Cimino is an associate professor at the Department of Information Engineering of the University of Pisa (Italy). His research lies in the areas of Information Systems and Artificial intelligence. He is (co-) author of about 70 scientific publications.

Gigliola Vaglini is full professor of Computer Engineering at “Dipartimento di Ingegneria della Informazione” of the University of Pisa. The main fields of her research activity are the specification and verification of concurrent and distributed systems, and the use of machine learning techniques for detecting malware in mobile systems and anomalies of industrial systems.