Continuous Control with a Combination of Supervised and Reinforcement Learning

Dmitry Kangin and Nicolas Pugeault
Computer Science Department,
University of Exeter
Exeter EX4 4QF, UK
{d.kangin, n.pugeault}@exeter.ac.uk

***Abstract*—Reinforcement learning methods have recently achieved impressive results on a wide range of control problems. However, especially with complex inputs, they still require an extensive amount of training data in order to converge to a meaningful solution. This limits their applicability to complex input spaces such as video signals, and makes them impractical for use in complex real world problems, including many of those for video based control. Supervised learning, on the contrary, is capable of learning on a relatively limited number of samples, but relies on arbitrary hand-labelling of data rather than task-derived reward functions, and hence do not yield independent control policies. In this article we propose a novel, model-free approach, which uses a combination of reinforcement and supervised learning for autonomous control and paves the way towards policy based control in real world environments. We use SpeedDreams/TORCS video game to demonstrate that our approach requires much less samples (hundreds of thousands against millions or tens of millions) comparing to the state-of-the-art reinforcement learning techniques on similar data, and at the same time overcomes both supervised and reinforcement learning approaches in terms of quality. Additionally, we demonstrate applicability of the method to MuJoCo control problems.**

I. Introduction

Recently published reinforcement learning approaches [1], [2], [3] have achieved significant improvements in solving complex control problems such as vision-based control. They can define control policies from scratch, without mimicking any other control signals. Instead, reward functions are used to formulate criteria for control behaviour optimisation. However, the problem with these methods is that in order to explore the control space and find the appropriate actions, even the most recent state-of-the-art methods require several tens of millions of steps before convergence [1], [4].

One of the first attempts to combine reinforcement and supervised learning can be attributed to [5]. Recently proposed supervised methods for continuous control problems, like [6], are known to rapidly converge, however they do not provide independent policies. Contrary to many classification or regression problems, where there exists a definitive ground truth, as it is presented in [7], [8], for many control problems an infinitely large number of control policies may be appropriate. In autonomous driving problems, for example, there is no single correct steering angle for every moment of time as different trajectories can be appropriate. However, a whole control sequence can be assessed according to objective measures: for example, by the average speed, time for completing a lap in a race, or other appropriate criteria. This situation is the same for other control problems connected with robotics, including walking [9] and balancing [10] robots, as well as in many others [11]. In these problems, also usually exist some criteria for assessment (for example, time spent to pass the challenge), which would help to assess how desirable these control actions are.

The problem becomes even more challenging if the results are dependent on the sequence of previous observations [12], e.g. because of dynamic nature of the problem involving speed or acceleration, or the difference between the current and the previous control signal.

In many real world problems, it is possible to combine reinforcement and supervised learning. For the problem of autonomous driving, it is often possible to provide parallel signals of the autopilot which could be used to restrict the reinforcement learning solutions towards the sensible subsets of control actions. Similar things can also be done for robotic control. Such real world models can be analytical, or trained by machine learning techniques, and may use some other sensors, which are capable to provide alternative information which enables to build the reference model (e.g., the model trained on LiDAR data can be used to train the vision based model). However, although there were some works using partially labelled datasets within the reinforcement learning framework [13], as far as we believe, the proposed problem statement, injecting supervised data into reinforcement learning using regularisation of Q-functions, is different from the ones published before. In [13], the authors consider the problem of robotic control which does not involve video data, and their approach considers sharing the replay buffer between the reinforcement learning and demonstrator data. In [14], the authors address the problem of sample efficiency improvement by using pretraining and state dynamics prediction. However, the proposed idea of pretraining is different from the one proposed here as the random positions and actions are used. Also, in contrast to the proposed method, which uses image input, the approach, proposed in [14], has only been tested on low-dimensional problems.

The novelty of the approach, presented in this paper, is given as follows:

1) a new regularised formulation optimisation, combining

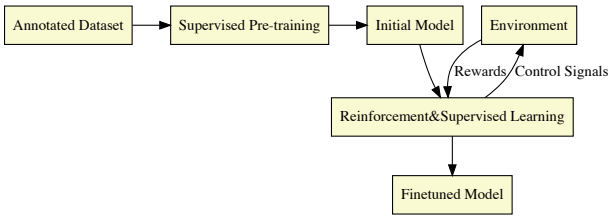


Fig. 1: The overall scheme of the proposed method

reinforcement and supervised learning, is proposed;
2) a training algorithm is formulated based on this problem statement, and assessed on control problems;
3) the novel greedy actor-critic reinforcement learning algorithm is proposed as a part of the training algorithm, containing interlaced data collection, critic and actor update stages.

The proposed method reduces the number of samples from millions or tens of millions, required to train the reinforcement learning model on visual data, to just hundreds of thousands, and is shown to converge to policies outperforming the ones achieved by both supervised and reinforcement learning approaches. This approach helps to build upon the performance of supervised learning by using reinforcement learning, which both provides the ability to learn faster than reinforcement learning and at the same time to improve policy using rewards, which is not possible when using just supervised training algorithms.

II. Proposed method

The overall idea of the method is shown in Fig. 1 and can be described as follows: to perform an initial approximation by supervised learning and then, using both explicit labels and rewards, to fine-tune it.

For supervised pre-training, the annotated dataset should contain recorded examples of control by some existing model. The aim of this stage is to mimic the existing control methods in order to avoid control behaviour of the trained model resulting in small rewards.

For supervised and reinforcement learning based fine-tuning, a pretrained model is used as an initial approximation. Contrary to the standard reinforcement learning approaches, the pre-trained model helps it to avoid control values resulting in small rewards right from the beginning. Also, for the control it is assumed that there is access to labels, which helps to divert the reinforcement learning model from spending its time on exploring those combinations of inputs and control signals, which are most likely not to provide meaningful solutions.

A. Supervised pretraining

Hereinafter, we consider the following problem statement for supervised pretraining. Let Z be the space of all possible input signals. To simplify the formalisation, we restrict ourselves to the image signals $z \in Z = \mathbb{R}^{m \times n}$, where m and n are the height and the width of the image, respectively. Such signals can form sequences of finite length $l > 0 : \langle z_1, z_2, \ldots, z_l \rangle \in Z^l$. We define an operator from the subsequences of real valued quantities to d-dimensional control signals $\pi(z_i, z_{i-1}, \ldots, z_{i-p+1}|\Theta_\pi) : Z^p \to \mathbb{R}^d$, where $i \in [p, l]$, $p \in \mathbb{N}$ is the number of frames used to produce the control signal, and Θ_π are the parameters of the operator $\pi(\cdot)$. We denote $c_i = \pi(z_i, \ldots, z_{i-p+1}|\Theta_\pi)$, $x_i = (z_i, z_{i-1}, \ldots, z_{i-p+1})$.

The problem is stated as follows: given the set of N sequences $\left\{\hat{z}^j \in Z^{l_j}\right\}_{j=1}^N$ and the set of corresponding signal sequences $\left\{\left\langle \hat{c}_i^j \in \mathbb{R}^d \right\rangle_{i=1}^{l_j}\right\}_{j=1}^N$, produced by some external control method called a 'reference actor', find the parameters Θ_π of the actor $\pi(\cdot|\Theta_\pi)$, which minimise a loss function (the used loss function is defined in Formula (7)).

B. Label-assisted reinforcement learning

The reinforcement learning method is inspired by the DDPG algorithm [3], however, it is substantially reworked in order to meet the needs of the proposed combination of supervised and reinforcement learning working in real time. First, the problem statement and basic definitions, necessary for formalisation of the method, need to be given.

In line with the standard terminology for reinforcement learning, we refer to a model, generating control signals, as an agent, and to control signals as actions. Also, as it is usually done for reinforcement learning problem statements, we assume the states to be equivalent to the observations. Initially, the model receives an initial state of the environment x_1. Then it applies the action c_1, which results in transition into the state x_2. The procedure repeats recurrently, resulting in sequences of states $X^n = \langle x_1, x_2, \ldots, x_n \rangle$ and actions $C^n = \langle c_1, c_2, \ldots, c_n \rangle$. Every time the agents performs an action it receives a reward $r(x_i, c_i) \in \mathbb{R}$ [3].

The emitted actions are defined by the policy π, mapping states into actions. In general [15], it can be stochastic, defining a probability distribution over the action space, however here the deterministic policy is assumed. For such a deterministic policy, one can express the discounted future reward using the following form of the Bellman equation for the expectation of discounted future reward $Q(\cdot, \cdot)$ [3]:

$$Q(x_i, c_i) = \mathbb{E}_{x_{i+1}} \left[r(x_i, c_i) + \gamma Q(x_{i+1}, \pi(x_{i+1})) \right], \quad (1)$$

where $\pi(\cdot)$ is an operator from states to actions, $\gamma \in [0, 1]$ is a discount factor.

$Q(\cdot, \cdot)$ is approximated by the critic neural network in line with the actor-critic approach. Similarly to [3], the proposed method uses a replay buffer to provide a training set.

In many of the state-of-the-art works on reinforcement learning, the actor's parameters are trained by maximisation of the critic's approximation of $Q(\cdot, \cdot)$, using gradient descent with gradients, obtained using the chain rule ([3], [15]):

$$\nabla_{\Theta_\pi} \pi(x|\Theta_\pi) \approx \mathbb{E}_\pi [\nabla_{\pi(x)} Q(x, \pi(x)|\Theta_Q) \nabla_{\Theta_\pi} \pi(x|\Theta_\pi)], \quad (2)$$

where Θ_Q and Θ_π are the trainable parameters of the actor and the critic, respectively. In [3], [15], the greedy optimisation

algorithm is not used as it might be impossible to collect diverse training sample for the critic as the actor would not be able to advance through the task. Instead, small steps in the gradient direction are applied to slightly modify the policy.

In the proposed method, contrary to many state-of-the-art methods, the optimisation is carried out in a greedy way, so that the steps of testing the current policy, updating the critic and the actor are interlaced. This is done in order to meet the requirements of real world scenarios, namely minimising the difference between the measurements per second rate in testing and training scenarios and providing the reasonable performance of the actor after the smallest possible number of epochs. In order to avoid deterioration of performance in the pretrained model (which would essentially lead to the number of steps comparable with the state-of-the-art reinforcement learning models), the regularisation is used to bring the parameters closer towards some pre-defined (reference) policy.

In the following derivations we use a restriction that $Q(\cdot,\cdot) \geq 0$, which, as one can easily see from Equation (1), will be true if the rewards are non-negative. We also assume (without loss of generality of further derivations) that the control signal is bounded between values (t_1, t_2), and t_1 and t_2 are not included into the appropriate operational values, in order to augment the training sample for the critic with the values $Q(x, t_1) = Q(x, t_2) = 0$ for every value x.

The optimisation problem is based on the regularised Q-function $f(x, \pi(x|\Theta_\pi))$:

$$F(\Theta_\pi) = \sum_{x \in X} f(x, \pi(x|\Theta_\pi)) \rightarrow \max_{\Theta_\pi}, \tag{3}$$

$$f(x, \pi(x|\Theta_\pi)) = \\ = Q(x, \pi(x|\Theta_\pi)) - \alpha Q(x, \hat{\pi}(x)) \rho(\pi(x|\Theta_\pi), \hat{\pi}(x)), \tag{4}$$

where $\alpha \geq 0$ is some coefficient, $\hat{\pi}(\cdot)$ is the reference actor, and $\rho(\cdot,\cdot)$ is a differentiable distance-like function. One can easily see that in the case $\alpha = 0$ it completely coincides with the reinforcement problem, and with large α values the problem becomes a standard supervised training one. The weight $Q(x, \hat{\pi}(x))$ is given to encourage the actor to follow the reference policy if the expected reward is high, and not to encourage much otherwise. Such type of penalisation gives the opportunity to establish a balance between the supervised approach, using only reference actor values, and reinforcement learning approach, which could help finding better policies comparing to the supervised approach and, in some cases, to the reference actor.

After differentiating this expression with respect to Θ_π one can see that

$$\nabla F(\Theta_\pi) = \sum_{x \in X} \nabla_{\Theta_\pi} \pi(x|\Theta_\pi) \times \\ \times \left\{ \nabla_\pi Q(x, \pi(x|\Theta_\pi)) - \alpha Q(x, \hat{\pi}(x)) \frac{\partial \rho(\pi(x|\Theta_\pi), \hat{\pi}(x))}{\partial \pi} \right\}. \tag{5}$$

As in [3], the update of the critic is carried out by solving the following optimisation problem:

$$\sum_{(x_i, c_i) \in (\hat{X}, \hat{C})} \Big[\hat{Q}^\pi(x_i, c_i|\Theta_Q) - r(x_i, c_i) - \\ -\gamma \hat{Q}^\pi(x_{i+1}, \pi(x_{i+1}|\Theta_\pi)|\Theta_Q) \Big]^2 \rightarrow \min_{\Theta_Q}, \tag{6}$$

where $\hat{X}$ and $\hat{C}$ are taken from the replay buffer, $\hat{Q}^\pi$ is the discounted reward function approximation. This equation is recurrent, and therefore the current target for training depends on previous training stages.

The training procedure is carried out in the way described in Algorithm 1. In this algorithm, NUM_EPOCHS denotes the maximum number of epochs during the training procedure.

Algorithm 1 The description of the training algorithm

Initialise and train the parameters Θ_π for the actor $\pi(x|\Theta_\pi)$ on a supervised dataset with observations and labels $\{X_S, C_S\}$
Initialise the parameters of the critic Θ_Q, $k = 1$
Initialise the empty replay buffer B
while $k \leq$ NUM_EPOCHS **do**
  Perform N_TESTING_EPISODES testing episodes of the length L_TESTING_EPISODES with the current parameters Θ_π, where the states x_i, actions c_i, reference actor actions c_i^{GT}, rewards $r_i = r(x_i, c_i)$ and subsequent states x_{i+1} are collected and put into the replay buffer B
  Perform N_GD_UPDATE_CRITIC iterations of gradient descent, according to the optimisation problem (6), in order to update the parameters of the critic Θ_Q; values x_i, x_{i+1}, c_i, r_i are taken from the replay buffer B
  Perform N_GD_UPDATE_ACTOR iterations of gradient descent for the optimisation problem (3) in order to update the parameters of the actor Θ_π; values x_i, $c_i^{GT} = \hat{\pi}(x_i)$ are taken from the replay buffer B
  $k = k + 1$
end while

As one can see from the algorithm, the 0-th epoch's testing episodes reflect the performance of the model with supervised pretraining only. This was done to assess the performance of the same model parameterisation during the epoch, as well as to exclude the problem when the control signal frequency (and hence performance) is affected by the optimisation time.

III. Results

To demonstrate the ability of the method to learn control signals, the TORCS [16] / SpeedDreams [17] driving simulator was used. In our experiments, the Simplix bot performed roles of both the baseline and the reference actor, which is a part of the standard package, was used. The outputs of the bot's steering angles were restricted between -0.25 and 0.25. The car name was Lynx 220. In the cases where car got stuck (average speed is less than one unit as measured by the simulator), the recovery procedure is provided by the

TABLE I: Parameters of the network

Parameter	Name in Algorithm 1	Value
Critic Training Steps/ Epoch	N_GD_UPDATE_CRITIC	12500
Actor Training Steps/ Epoch	N_GD_UPDATE_ACTOR	12500
Testing episode length	L_TESTING_EPISODES	3000
Testing episodes per epoch	N_TESTING_EPISODES	5
Actor learning rate		$1 \cdot 10^{-4}$
Critic learning rate		$1 \cdot 10^{-5}$
Supervised learning rate		$1 \cdot 10^{-4}$
Supervised momentum		0.9
Soft update coefficient		0.1
Discount factor γ	See Equation 1	0.9
Actor FC1 size	See Fig. 2	2002
Actor FC2 size	See Fig. 2	2002
Actor FC3 size	See Fig. 2	1
Critic FC1 size	See Fig. 2	2005
Critic FC2 size	See Fig. 2	2005
Critic FC3 size	See Fig. 2	1

bot. The time and rewards for the recovery are excluded from consideration. The reward is defined as the current car speed measured by the simulator. The assessment has been carried out on a single computer, using NVIDIA GeForce GTX 980 Ti graphical processor on Ubuntu Linux operating system. The model parameters are described in section III-A.

Another aspect to be considered is how the proposed approach would work for other well-known control scenarios. Section III-D discusses the performance of the method for MuJoCo environments [18] using OpenAI Baselines package [19].

A. Parameterisation of the model

The parametrisation for the experiments is given in Table I; the parameters' verbal description is augmented with the names referring to Algorithm 1.

For supervised only pretraining of the actor network, the Momentum algorithm is used [20]; for the rest of the stages, the Adam algorithm is used [21]. The proposed algorithm has been implemented in Python using TensorFlow framework [22]. For the stage of supervised pretraining, in order to improve convergence of the model at the initial stage, the additional soft update coefficient was introduced for exponential smoothing of the parameters of the network during gradient descent optimisation. In all the experiments, pretraining data contained up to a few thousand of states and served only to provide the agent some initial policies conforming with the behaviour of the reference actor.

B. Network architecture

The network architecture, used for the proposed actor-critic approach, is shown in Fig. 2. For supervised pretraining, the dataset has been collected from the SpeedDreams simulator, combining the sequences of images and the corresponding control values produced by the bot. Using this dataset, the InceptionV1 network [23] is first fine-tuned from the Tensorflow Slim implementation [24] on the collected dataset, mapping each single image directly into the control signals. The last (classification) layer of the network is replaced by a fully

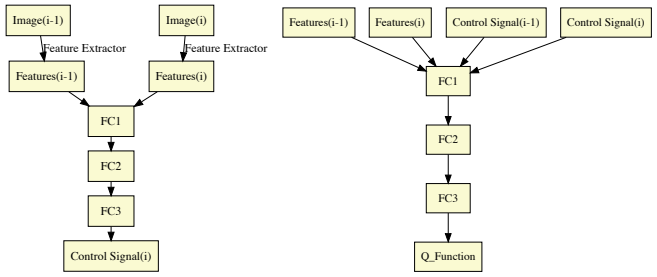


Fig. 2: The scheme of the actor (left) and critic (right) networks

connected layer of the same size (1001) as the preceding logits layer; this layer is referred to as a feature extraction layer as it serves as an input space for the actor-critic model. Such an approach is used in order to avoid expensive storage of images in the replay buffer, as well as circumvent the challenge of training deep architectures within the reinforcement learning setting. For two input images, the extracted features are concatenated into the fully connected layer FC1, followed by (also fully connected) layers FC2 and FC3. Each of the actor and critic layers except the last, FC3, contains ReLU non-linearity [25]. The actor's last layer, responsible for control signals production, with dimensionality d of the control signal is followed by the $\tanh$ non-linearity, which restricts the output between the boundary values $(t_1, t_2) = (-1, 1)$; the last layer of the critic has no non-linearity.

For supervised fine-tuning and pretraining stages, we use the following loss function:

$$l(X, C) = \sum_{x \in X, c \in C} \frac{|\pi(x) - c|}{|c| + \epsilon}, \tag{7}$$

where ϵ is a reasonably small constant (with respect to c, we use $\epsilon = 10^{-2}$), $\pi(x)$ is an operator, transforming input vectors x to the control signals, c is the reference actor control signal. This loss is chosen in order to prevent the model from overfitting to larger reference actor control signals.

Similarly, the distance-like function ρ in Formula (4) is defined as:

$$\rho(\pi, \hat{\pi}) = \frac{(\pi - \hat{\pi})^T (\pi - \hat{\pi})}{|\hat{\pi}|^2 + \epsilon}, \tag{8}$$

where ϵ is a reasonably small constant (we use $\epsilon = 10^{-4}$).

To implement the formalisation, described in section II-A, we use a siamese network architecture (see [26]), given in Fig. 2. The features, calculated by the fine-tuned network, are submitted for the current and the previous frame ($p = 2$ in terminology of section II-A). It is done in order to take into account the dynamic state of the environment, including speed. Also we need to mention that the previous control signal is submitted as an input for the critic (for the initial video frame in each sequence, we assume that the frames and control signals are the same for the current and the previous frames).

TABLE II: Simplix bot rewards

Episode 1	Episode 2	Episode 3	Episode 4	Episode 5	Average
81315	81779	79481	81564	81931	81214

TABLE III: Maximum sum-of-rewards values for different parameters α (for $\alpha = 0$ these values are obtained during supervised pretraining)

α	$\max_R$	$100\% \times \frac{\max_R}{\max_{R_{bot}}}$	Epoch	$\max_{\overline{R}}$	$100\% \times \frac{\max_{\overline{R}}}{\overline{R_{bot}}}$	Epoch
100	75512	92.16	9	72180	88.88	10
0.25	75896	**92.63**	17	73339	90.30	14
0.1	75867	92.60	32	73381	**90.35**	15
0.05	75379	91.97	27	71670	88.25	23
0.01	67766	82.71	20	65806	81.03	20
0	62725	76.56	1	58469	71.99	1

C. Experiments

Table III and Fig. 3 record rewards achieved for different values of the parameter α. When $\alpha = 0$, we rely solely on reinforcement learning; high values of α enforce a stronger bias towards supervised labels rather than improving the policy by reinforcement learning.

In Fig. 3, the scatter points depict total rewards during each of the testing episodes, and the curves show mean total rewards for each epoch over all testing episode rewards. The total reward is calculated as an arithmetic sum of the rewards during one training episode. The left figure shows the results for all tested parameters, while the right one shows the comparison between the best performing one and the one with the largest value of α (i.e. the closest to supervised active learning). The performance of the pretrained model corresponds to the first epoch in the graph. The shaded area in the right figure shows the standard deviation of the testing episodes performance during the epoch. One can see from the figure that smaller values of the coefficient α tend to yield worse performance; however, it may be attributed to longer convergence time as it implies more reliance on reinforcement learning exploration. At the same time, unlimited increase of the coefficient α does not help gaining further performance improvements. Even more, one can see from the right figure that after certain point the curve for $\alpha = 100$ slowly declines; we suggest that it could be caused by overfitting to the bot values. We also see that for $\alpha = 0$, when the supervised data is used only during the pre-training stage, the performance is much lower than for the rest of the graphs.

Table III shows the total rewards for different values of parameter α. The value $\max_R$ shows the maximum total reward over one testing episode during all the training time (corresponds to the highest scatter point in Fig. 3 for a given parameter α), the value $\max_{\overline{R}}$ shows the maximum mean total reward, where the mean is calculated for each epoch over all testing episode total rewards (corresponds to the highest point of the curve in Fig. 3 for a given parameter α).

In order to compare the rewards shown in these graphs and tables, we have also measured them the Simplix bot in the same conditions (frame per second rate) as the proposed algorithm. The total rewards for the Simplix bot, available in the SpeedDreams/TORCS simulator, are given in Table II. The mean value of these rewards is $\overline{R_{bot}} = 81214.07$. The maximum value achieved by the bot is $\max_{R_{bot}} = 81931.15$. The percentage of the proposed algorithm's rewards with respect to the bot one, in average and for the best performance, is also given in Table III.

One of the notable things is the dramatic decrease in the number of samples: from millions [3] or tens of millions [1] measurements for standard reinforcement learning techniques [3] to just hundreds of thousands (15000 per training epoch, several tens of training epochs). For some of the reinforcement learning algorithms, trained on a problem of driving in a simulator, only some realisations were able to finish the task [3], and for those methods, which report solving similar problems by reinforcement learning only, the reported performance constitutes $75 - 90\%$ [4], while we achieve up to 92.63% of the bot's performance as reported in Table III. The graphs in Fig. 4 and 5 illustrate the difference between the unregularised Q-function $Q(x, \pi(x))$ and its regularised counterpart $f(x, \pi(x))$ (see Equation 4). The images on the top of the figures, recorded during the testing episode of the first epoch (directly after the supervised pretraining), have been used for computation of the Q-values. The graphs below these images show the evolution of the Q-function (and its regularised version f) across the epochs. One can see that, as expected, the regularised Q-function is steeper than the unregularised one as it penalises the values which are too far away from the reference actor control signal. The difference between Fig. 4 and 5 shows the importance of the factor $Q(x, \hat{\pi}(x))$ in Equation 4: the penalty for not following the control signal is less, if the expected reward is smaller. It makes the algorithm explore values further from the bot's values, if it does not provide high expected reward.

D. Experiments with regularised DDPG on MuJoCo environments

The experiments with MuJoCo environment [18] demonstrate applicability of the proposed idea of regularising Q-values with supervised learning to other well known control problems, while showing that in certain cases the approach can outperform the original reference actor. For this purpose, we have added the proposed Q-function regularisation to the standard DDPG algorithm with the implementation, network architecture and parameters, taken from OpenAI Baselines [19]. L^2 distance has been used as ρ for the equation (4). Also, in order to maintain the correctness of regularisation assumptions, as the condition $Q \geq 0$ is not met for some low reward values in MuJoCo, $Q(x, \hat{\pi}(x))$ was substituted by $\max(0, Q(x, \hat{\pi}(x))$. The reference actors were obtained by pretraining actors by standard DDPG algorithm.

The results of these experiments are given in Fig. 6. In every graph, the black lines show the performance of the pretrained

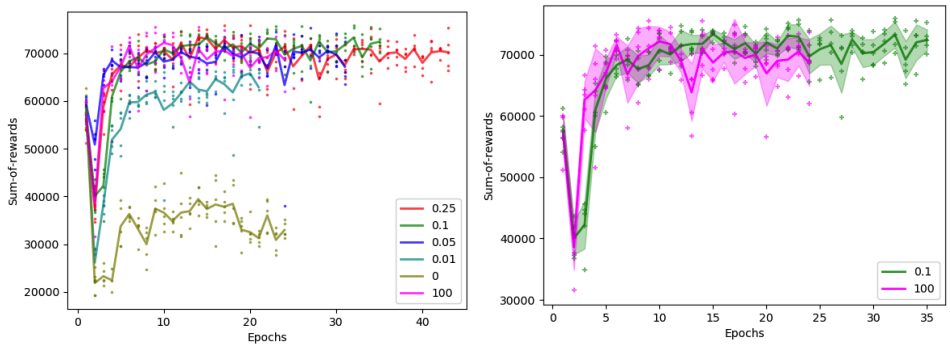


Fig. 3: Sum-of-rewards for different parameters α

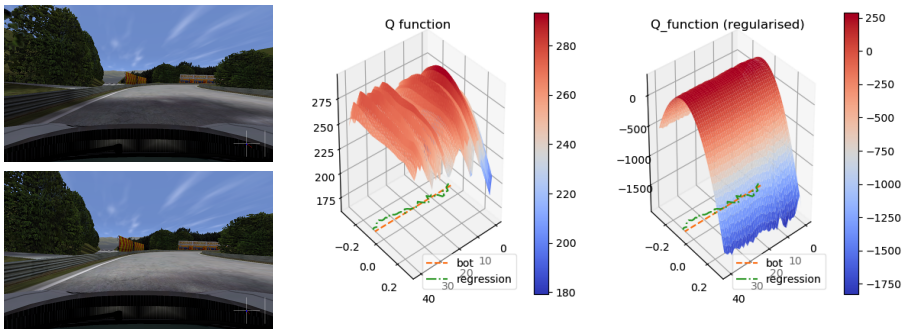


Fig. 4: Regularised and non-regularised Q-function values, high reward, $\alpha = 0.25$; the epochs' indices are shown on the right axis, and the values of $\pi(\cdot)$ are shown on the left axis

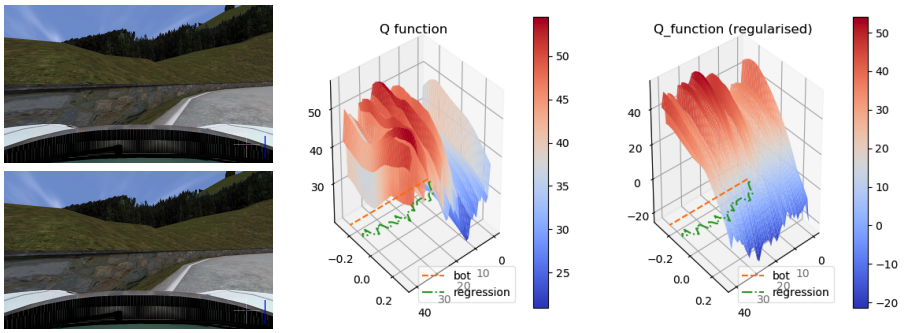


Fig. 5: Regularised and non-regularised Q-function values, low reward, $\alpha = 0.25$; the epochs' indices are shown on the right axis, and the values of $\pi(\cdot)$ are shown on the left axis

reference actor. These experiments are aimed to compare the following three cases:

1) the original DDPG algorithm (referenced as $\alpha = 0$)
2) the DDPG algorithm with fixed regularisation coefficient, $\alpha = 0.1$
3) the DDPG algorithm with exponential decay, initial value of α is 1, the value is decayed with the coefficient 0.01 every $20,000$ timesteps.

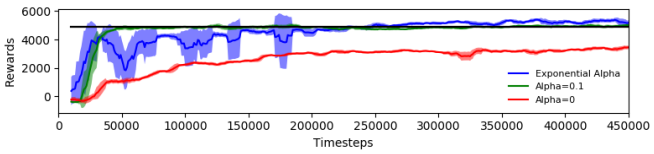
For the HalfCheetah scenario, one can see that the model with the fixed regularisation coefficient can easily reach the pretrained value but then lags behind the algorithm with exponential decay. The exponential decay algorithm, in contrary, takes advantage of the reference actor performance and then gradual decay of the regularisation enables it to explore values further from the reference actor. These results could suggest that in certain cases the regularisation can prevent the model of further exploration beyond the reference actor performance.

For the Hopper scenario, the peak in the performance of original DDPG algorithm beyond the reference actor baseline near step $270,000$ suggests that the original DDPG algorithm may be unstable for this task, which also holds for DDPG with exponential decay as the regularisation coefficient becomes negligibly small. At the same time one can see that the model with the fixed regularisation coefficient can reach performance

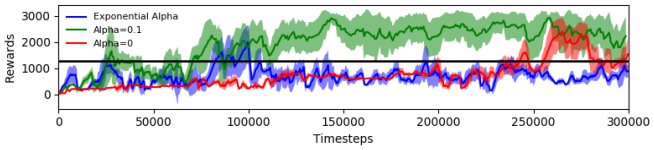
beyond the reference actor.

It could be concluded from the graphs for the InvertedDoublePendulum task that convergence depends in this case on the initial value of parameter α. The larger initial value for the DDPG with exponential decay appears to give better results due to heavier reliance on the supervised part. Importantly, all versions of the method are able to maintain stable performance after the initial training episode.

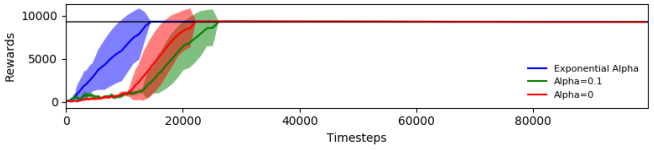
For the Swimmer scenario, the exponential α setting allowed to go for some period of time beyond the reference actor baseline; at the same time, the version with $\alpha = 0.1$, while not beating the reference actor, shows smaller variance than the original algorithm, and most of the time exposes better average performance than the original DDPG method.



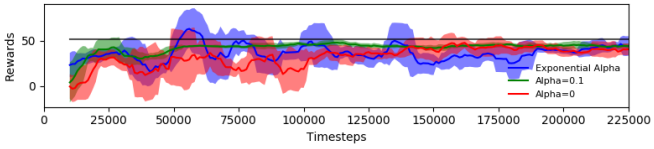
(a) HalfCheetah



(b) Hopper



(c) InvertedDoublePendulum



(d) Swimmer

Fig. 6: Results in MuJoCo environments, top to bottom: HalfCheetah, Hopper, InvertedDoublePendulum, Swimmer. The average reference actor performance is shown by black horizontal bar, $\alpha = 0$ corresponds to the original DDPG algorithm

IV. Conclusion

The proposed method shows dramatic improvement in the number of samples for video data (down to just several hundred thousand) comparing to the reinforcement learning methods, as well as improves the performance on simulated robotic tasks comparing to both supervised and reinforcement learning. We believe that such approach, combining reinforcement and supervised learning, could help to succeed in the areas of complex spaces where the state-of-the-art reinforcement learning methods are not working yet, as well as towards practical usage for real world models such as autonomous cars or robots.

However, there are still a few limitations of the proposed method. First, it still requires label data through all the course of training. We believe that in the future work it should be possible to reduce usage of training data to a limited number of labelled episodes. Such decrease of the training data could benefit to the range of practical tasks solvable by the proposed approach.

V. Acknowledgement

The authors are grateful for the support by the UK Engineering and Physical Sciences Research Council (EPSRC) project DEVA EP/N035399/1.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey." *Reinforcement Learning, Springer Berlin Heidelberg*, pp. 579–610, 2012.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] V. Mnih, A. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International Conference on Machine Learning*, pp. 1928–1937, 2016.

[5] M. T. Rosenstein, A. G. Barto, J. Si, A. Barto, W. Powell, and D. Wunsch, "Supervised actor-critic reinforcement learning," *Handbook of Learning and Approximate Dynamic Programming*, pp. 359–380, 2004.

[6] T. Welschehold, C. Dornhege, and W. Burgard, "Learning manipulation actions from human demonstrations," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3772–3777.

[7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." *AAAI*, pp. 4278–4284, 2017.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[9] K. Li and R. Wen, "Robust control of a walking robot system and controller design." *Procedia Engineering*, vol. 174, pp. 947–955, 2017.

[10] N. Esmaeili, A. Alfi, and H. Khosravi, "Balancing and trajectory tracking of two-wheeled mobile robot using backstepping sliding mode control: Design and experiments," *Journal of Intelligent & Robotic Systems*, pp. 1–13, 2017.

[11] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286.*, 2017.

[12] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[13] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[14] C. W. Anderson, M. Lee, and D. L. Elliott, "Faster reinforcement learning after pretraining deep networks to predict state dynamics," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–7.

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

[16] E. Espi, C. Guionneau, and B. W. et al., "Torcs, the open racing car simulator," http://torcs.sourceforge.net, accessed in January 2018.

[17] SpeedDreams, "an open motorsport sim," http://www.speed-dreams.org, accessed in January 2018.

[18] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[19] P. Dhariwal, C. Hesse, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Openai baselines," https://github.com/openai/baselines, 2017.

[20] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks : The Official Journal of the International Neural Network Society*, vol. 12(1), p. 145151, 1999.

[21] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, E. Dumitru, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[24] N. Silberman and S. Guadarrama, "Tf-slim: A high level library to define complex models in tensorflow," https://research.googleblog.com/2016/08/tf-slim-high-level-library-to-define.html, 2016, accessed in January 2018.

[25] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[26] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," *European Conference on Computer Vision, Springer International Publishing*, vol. 18, pp. 850–865, 2016, October.