A Cat-Like Robot Real-Time Learning to Run

Paweł Wawrzyński

Warsaw University of Technology, Poland

Abstract. Actor-Critics constitute an important class of reinforcement learning algorithms that can deal with continuous actions and states in an easy and natural way. In their original, sequential form, these algorithms are usually to slow to be applicable to real-life problems. However, they can be augmented by the technique of experience replay to obtain a satisfying speed of learning without degrading their convergence properties. In this paper experimental results are presented that show that the combination of experience replay and Actor-Critics yields very fast learning algorithms that achieve successful policies for nontrivial control tasks in considerably short time. Namely, a policy for a model of 6-degree-of-freedom walking robot is obtained after 4 hours of the robot's time.

Keywords: reinforcement learining, actor-critics, experience replay, neural networks.

1 Introduction

Reinforcement learning (RL) addresses the problem of an agent that optimizes its reactive policy in a poorly structured and initially unknown environment [9]. Algorithms developed in this area can be viewed as computational processes that transform observations of states, actions and rewards into policy parameters. Several important RL algorithms, such as Q-Learning [10] and Actor-Critic methods [2,5,6,3], process the data sequentially. Each single observation is used for adjusting the algorithms' parameters and then becomes unavailable for further use. We shall call such methods *sequential*. They are based on a common assumption that RL applications to real-world learning control problems require large amounts of data which cannot be kept in a limited amount of memory assigned to the algorithm.

Sequential algorithms do not exploit all the information contained in the data and are known to require a large number of environment steps to obtain a satisfying policy. Usually, this number is large enough to make the learning process detrimental to any real device whose control policy we would hope to optimize by means of RL. However, there are other, *non-sequential* methods that require much fewer environment steps to obtain a policy of the same quality. They achieve this at the cost of collecting data and some extensive processing thereof. This distinction between sequential and non-sequential algorithms should not be confused with the distinction between online and offline algorithms. Here, we are interested *only* in online algorithms which improve the policy as the agent-environment interaction proceeds. Online sequential and non-sequential algorithms differ in the way of using the available computational power during the interaction.

One of the approaches to design of non-sequential algorithms consists in adding some non-sequential data processing to a sequential algorithm. The modified algorithm collects historical experiences (observations of states, actions and rewards) and applies to them the operations of the original sequential algorithm as if they have just taken place. This idea of repeating many similar operations to the same event, called *experience replay* [8,7,4], was popular a few years ago but has received little attention recently. Unfortunately, experience replay is not automatically applicable to an arbitrary RL algorithm. In particular, it cannot be directly combined with on-policy methods, i.e., those based on the assumption that the actions producing data for policy improvements are drawn from the current policy. Consequently, the same experience cannot be applied many times to adjust a continuously changing policy.

In this paper we present an experimental study on experience replay and an Actor-Critic-type learning algorithm combined in a fashion introduced in [12]. In the study we obtain a policy of an emulated cat-like robot called Half-Cheetah [11]. The robot is a kinematic string with 6-degrees of freedom. Its state space is 31 dimensional. The learning goal is to make Half-Cheetah run as fast as possible. The objective is obtained within 4 hours of Half-Cheetah time.

The paper is organized as follows. In Sec. 2 the problem of our interest is defined along with the class of algorithms that encompasses sequential Actor-Critics. Section 3 shows how to estimate improvement directions in the policy parameters' space using the data from the preceding state transition and to accelerate a sequential algorithm by combining these estimators with experience replay. The experimental study is presented in Sec. 4.

2 Problem Formulation

We will consider the standard RL setup [9]. A Markov Decision Process (MDP) defines a problem of an agent that observes its state s_t in discrete time $t = 1, 2, 3, \dots$, performs actions a_t, receives rewards r_t and moves to other states s_{t+1}. A particular MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, P_s, r \rangle$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively; $\{P_s(\cdot|s, a) : s \in \mathcal{S}, a \in \mathcal{A}\}$ is a set of state transition distributions; we write $s_{t+1} \sim P_s(\cdot|s_t, a_t)$ and assume that each P_s is a density. Each state transition generates a *reward*, $r_t \in \Re$. Here we assume that each reward is depends deterministically on the current action and the next state, $r_t = r(a_t, s_{t+1})$.

Actions are generated according to a policy, π, which is a family of distributions parameterized by the state and a *policy vector* $\theta \in \Re^{n_\theta}$, namely $a_t \sim \pi(\cdot\,; s_t, \theta)$. The objective of reinforcement learning is to optimize θ to make the policy maximize future rewards. This goal may be strictly specified in various ways. We may require the policy to maximize the average reward or to maximize the sum of future discounted rewards expected in each state.

Below we analyze a large class of the existing sequential RL methods suitable for policy determination in simulations and propose a way of their acceleration based on a more extensive data processing. Our intention is to design methods that obtain a satisfying policy after a much smaller amount of agent time but not necessarily after a smaller amount of computation. The control efficiency should be maximized within short period of learning to keep the controlled machine from being damaged by too many wrong actions.

Sequential Actor-Critics. Actor-Critics [2,5,6,3] constitute probably the most efficient and the most theoretically developed class of reinforcement learning algorithms. Let us analyze an example of methods that will be of interest for us: the algorithm presented in [5], quoted in the table beside, and called here the Basic Actor-Critic. The actor is represented here, as usually, by the parameterized policy π. The critic is represented by the approximator $\bar{V}(s; \upsilon)$ parameterized by the critic vector $\upsilon \in \Re^{n_\upsilon}$. The critic approximates the value function V^π, that is for a given policy, π, equal to the sum of future discounted rewards expected in a given state, namely $V^\pi(s) = E\left(\sum_{i\geq 0} \gamma^i r_{t+i} \middle| s_t = s, \pi\right)$, where $\gamma \in (0, 1)$ is a discount factor. The values β_t^θ and β_t^υ are the step-sizes: they are positive reals decreasing with growing t. Also, they should satisfy the standard stochastic approximation conditions: $\sum_{t\geq 1} \beta_t = \infty, \sum_{t\geq 1} \beta_t^2 < \infty$.

Algorithm 1. The Basic Actor-Critic. $\gamma \in (0, 1)$ is a discount factor, $\lambda \in [0, 1]$, $\bar{V}$ is the value function approximator (the critic) parameterized by vector υ. β_t^θ and β_t^υ are the step-sizes.

```
0: Set $y_\upsilon = 0, y_\theta = 0, t := 1$. Initialize $\theta$ and $\upsilon$.
1: Draw the action, $a_t \sim \pi(\cdot\,; s_t, \theta)$.
2: Execute $a_t$, evaluate the next state $s_{t+1}$ and the reward $r_t$.
3: Calculate the *temporal difference* of the form
4:     $d_t(\upsilon) = r_t + \gamma \bar{V}(s_{t+1}; \upsilon) - \bar{V}(s_t; \upsilon)$.
5: Adjust the actor:
6:     $y_\theta := (\gamma\lambda) y_\theta + \beta_t^\theta \nabla_\theta \ln \pi(a_t; s_t, \theta)$
7:     $\theta := \theta + y_\theta d_t(\upsilon)$
8: Adjust the critic:
9:     $y_\upsilon := (\gamma\lambda) y_\upsilon + \beta_t^\upsilon \nabla_\upsilon \bar{V}(s_t; \upsilon)$
10:    $\upsilon := \upsilon + y_\upsilon d_t(\upsilon)$.
11: Set $t := t + 1$ and repeat from Point 1.
```

Below, we provide a simplistic, appealing to intuition, analysis of this algorithm. We show it as working in the following way: It increases the probability of a given action a_t if it turns out to lead to higher rewards than expected in state s_t. If the action turns out to lead to smaller rewards, its probability is decreased. Namely, let us consider the total adjustment of the policy vector θ during the work of the algorithm. To this end, we analyze the value of y_θ by the end of the algorithm's loop. It can be seen that y_θ is then equal to

$$y_\theta = \sum_{k=0}^{t-1} (\gamma\lambda)^k \beta_{t-k}^\theta \nabla_\theta \ln \pi(a_{t-k}; s_{t-k}, \theta).$$

Therefore, the total adjustment is equal to

$$\varDelta\theta = \sum_{t>0} d_t(\upsilon) \sum_{k=0}^{t-1} (\gamma\lambda)^k \beta_{t-k}^\theta \nabla_\theta \ln \pi(a_{t-k}; s_{t-k}, \theta)$$

By changing the summation order we obtain

$$\varDelta\theta = \sum_{t>0} \beta_t^\theta \nabla_\theta \ln \pi(a_t; s_t, \theta) \sum_{k\geq 0} (\gamma\lambda)^k d_{t+k}(\upsilon). \quad (1)$$

We can see that an element of the above sum attributes to the state s_t the total adjustment of the policy vector that the visit in this state induces. The adjustment is equal to the product of a vector and a sum of scalars. The vector, $\nabla_\theta \ln \pi$, defines the direction in which θ must be modified to change the probability of action a_t in state s_t. The sum, $\sum_{k\geq 0}$, determines whether the action a_t leads to higher rewards than expected in state s_t (then the sum is positive and the probability of a_t is increased) or a_t leads to smaller rewards than expected in s_t (the sum is negative and the probability of a_t is decreased).

The critic training. A similar analysis reveals the compact form of the total adjustment of the critic vector. Namely, at the end of the algorithm's loop, the vector y_υ is equal to

$$y_\upsilon = \sum_{k=0}^{t-1} (\gamma\lambda)^k \nabla_\theta \ln \pi(a_{t-k}; s_{t-k}, \theta).$$

Therefore, the total adjustment of the critic vector is equal to

$$\varDelta\upsilon = \sum_{t>0} \beta_t^\upsilon d_t(\upsilon) \sum_{k=0}^{t-1} (\gamma\lambda)^k \nabla_\upsilon \bar{V}(s_{t-k}; \upsilon)$$

By changing the summation order we obtain

$$\varDelta\upsilon = \sum_{t>0} \beta_t^\upsilon \nabla_\upsilon \bar{V}(s_t; \upsilon) \sum_{k\geq 0} (\gamma\lambda)^k d_{t+k}(\upsilon). \quad (2)$$

We can see that an element of the above sum attributes to the state s_t the total adjustment of the critic vector that the visit in this state induces. In order to understand the character of this adjustment, one may notice that the inner sum in Eq. (2) is the same as the inner sum in Eq. (1) expressing how large future turned out to be in comparison to expected in state s_t. Hence, the critic training consists in increasing $\bar{V}(s_t; \upsilon)$ when actural rewards turned to be higher than this value, and decreasing otherwise.

Generalization. In general, we will consider sequential actor-critic-type algorithms characterized by the following features:

1. Actions are generated by a stationary policy (actor) i.e., a distribution π parameterized by state s_t and the policy vector $\theta \in \Re^{n_\theta}$: $a_t \sim \pi(\cdot\,; s_t, \theta)$.
2. A visit in state s_t causes a modification of the policy vector θ by a product $\beta_t^\theta \widehat{\phi}_t$, where $\widehat{\phi}_t$ on average indicates the direction in which θ assures larger future rewards expected in state s_t whereas $(\beta_t^\theta, t = 1, 2, \dots)$ is a vanishing sequence of step-sizes.

3. The algorithm may compute $\widehat{\phi}_t$ with the use of an auxiliary parameters $\upsilon \in \Re^{n_\upsilon}$. A visit in state s_t results in a modification of υ by a vector $\beta_t^\upsilon \widehat{\psi}_t$, where $\widehat{\psi}_t$ on average points into the direction where υ assures better quality of $\widehat{\phi}_t$ whereas $(\beta_t^\upsilon, t = 1, 2, \dots)$ is a vanishing sequence of step-sizes.
4. The vectors $\widehat{\phi}_t$ and $\widehat{\psi}_t$, different from one another, are of the same form
$$G_t(\theta, \upsilon) \sum_{k \geq 0} (\alpha\rho)^k z_{t,k}(\theta, \upsilon) \tag{3}$$
where G_t is a vector defined by s_t and a_t, $\alpha \in [0, 1)$, $\rho \in [0, 1)$, and $z_{t,k} \in \Re$ is defined by $s_{t+k}, a_{t+k}, r_{t+k}, s_{t+k+1}$, and possibly a_{t+k+1}.

In the Basic Actor-Critic algorithm mentioned above we have
$$\widehat{\phi}_t = \nabla_\theta \ln \pi(a_t; s_t, \theta) \sum_{k \geq 0} (\gamma\lambda)^k d_{t+k}(\upsilon), \quad \widehat{\psi}_t = \nabla_\upsilon \bar{V}(s_t; \upsilon) \sum_{k \geq 0} (\gamma\lambda)^k d_{t+k}(\upsilon),$$
which means that both $\widehat{\phi}_t$ and $\widehat{\psi}_t$ are of the form (3) with $\gamma\lambda = \alpha\rho$, and
$$G_t(\theta, \upsilon) = \nabla_\theta \ln \pi(a_t; s_t, \theta), \quad z_{t,k}(\theta, \upsilon) = d_{t+k}(\upsilon)$$
for the actor, while for the critic those are equal to
$$G_t(\theta, \upsilon) = \nabla_\upsilon \bar{V}(s_t; \upsilon), \quad z_{t,k}(\theta, \upsilon) = d_{t+k}(\upsilon).$$

Important algorithms that also fit into the discussed schema are the actor-critics presented in [6,3] and OLPOMDP [1].

Let us analyze the average direction of $\widehat{\phi}_t$ and $\widehat{\psi}_t$. Namely, let ϕ be a function defined as
$$\phi(s, \theta, \upsilon) = E_{\theta,\upsilon,\beta} \left(\widehat{\phi}_t | s_t = s \right). \tag{4}$$
The definition of ϕ is based on the assumption that θ, υ, and the step-sizes remain constant when $\widehat{\phi}_t$ is calculated. In fact, they slightly vary and each $\widehat{\phi}_t$ is in fact a biased estimator of $\phi(s_t, \theta, \upsilon)$ for θ and υ used at time t. However, this bias is small and since the dynamics of the parameters decreases in time, the bias asymptotically vanishes. The average $\phi(s, \theta, \upsilon)$ weighed by the steady-state distribution defines the direction of the drift of the policy vector.

The drift of υ may be analyzed in a similar way. Namely, let ψ be a function defined as
$$\psi(s, \theta, \upsilon) = E_{\theta,\upsilon,\beta} \left(\widehat{\psi}_t | s_t = s \right). \tag{5}$$
As above, the definition of ψ requires that θ, υ, and the step-sizes remain constant during the time when $\widehat{\psi}_t$ is computed. The drift of υ is defined by the average $\psi(s, \theta, \upsilon)$ weighed by the steady-state distribution. The usual role of the drift of the auxiliary parameter is to move it toward the point $\upsilon^*(\theta)$ such that the average $\phi(s, \theta, \upsilon^*(\theta))$ approximates either a policy gradient or a natural policy gradient. Hence, adjustments of θ ultimately lead to policy improvement.

3 Experience Replay

The main idea analyzed in this paper is to apply to the agent's experience the same processing as a sequential actor-critic-type algorithm would, yet more intensively. A generic algorithm augmented by experience replay is presented in the table below.

After each instant t, the original sequential algorithm estimates $\phi(s_t, \theta, v)$ i.e., the direction of policy improvement, and adjusts the policy vector θ along the estimate. Within each instant t, the modified algorithm repeatedly draws one of the recently visited states, s_i, estimates $\phi(s_i, \theta, v)$, and modifies the policy vector along the estimate. Essentially both algorithms achieve the same goal but (i) the modified one does it more intensively and (ii) it employs experience gathered after visiting state s_i to adjust various policies characterized by different policy vectors. The auxiliary vector v undergoes similar operations in both algorithms.

Algorithm 2. Actor-Critic with Experience Replay. Estimators mentioned in Steps 6 and 7 are based on the data in a database.

```
 0: $t := 1$. Initialize $\theta$ and $v$.
 1: Draw and execute an action, $a_t \sim \pi(\cdot\,; s_t, \theta)$.
 2: Register the tuple $\langle s_t, a_t, \theta, r_t, s_{t+1} \rangle$ in the
    database.
 3: Make sure only $N$ most recent tuples remain
    in the database.
 4: Repeat $\nu(t)$ times:
 5:     Draw $i \in \{t-N+1, t-N+2, \ldots, t\}$.
 6:     Adjust $\theta$ along an estimator of $\phi(s_i, \theta, v)$:
 7:         $\theta := \theta + \beta^\theta_t \widehat{\phi}^r_i(\theta, v)$.
 8:     Adjust $v$ along an estimator of $\psi(s_i, \theta, v)$:
 9:         $v := v + \beta^v_t \widehat{\psi}^r_i(\theta, v)$.
10: Assign $t := t + 1$ and repeat from Step 1.
```

Because the policy vector is constantly changing, each time its adjustment is performed, it is computed on the basis of the new values of θ and v. The intensity of replaying, $\nu(t)$, must be bounded for the sake of correctness of the algorithm. It is also limited by the computation power available during the agent–environment interaction. $\nu(t)$ should be additionally limited for small t to prevent many recalculations of few tuples in the database and to avoid overtraining.

Designing the estimators of ϕ and ψ for Steps 7 and 9 of Algorithm 3 we have to guarantee that their variance is bounded and their bias asymptotically vanishes. This is the only way for the algorithm to preserve the limit properties of the original sequential method.

Let $b > 1$, θ_{i+j} be the policy vector applied to generate a_{i+j}, and K be drawn independently from $Geom(\rho)$, the geometric distribution[1] with parameter $\rho \in [0, 1)$. Also, let χ be equal to 0 if $z_{t,k}$ is not explicitly defined by a_{t+k+1} (as in the basic AC), and to 1 otherwise (as in AC of [6]). We introduce the *randomized-truncated estimators* $\widehat{\phi}^r_i(\theta, v)$ and $\widehat{\psi}^r_i(\theta, v)$ of the same generic form

$$\sum_{k=0}^{K} G_i(\theta, v)\alpha^k z_{i,k}(\theta, v) \min\left\{ \prod_{j=0}^{k+\chi} \frac{\pi(a_{i+j}; s_{i+j}, \theta)}{\pi(a_{i+j}; s_{i+j}, \theta_{i+j})}, b \right\}. \quad (6)$$

[1] That is, random variable K of values in $\{0, 1, 2, \ldots\}$ has distribution $Geom(\rho)$, iff $P(K = m) = (1-\rho)\rho^m$ for nonnegative integer m.

where $\widehat{\phi}_i^r$ is defined by those G and z defining $\widehat{\phi}_t$, and $\widehat{\psi}_i^r$ is defined by those G and z that define $\widehat{\psi}_t$. We can see that (6) closely resembles the original form (3) of $\widehat{\phi}_t$ and $\widehat{\psi}_t$ with two important differences. First, the infinite sum is replaced by the finite one with the appropriately designed random limit. Second, truncated density ratios are introduced in order to compensate for the fact that the current policy is different than the one that generated the actions $a_t, a_{t-1}, \ldots$ contained in the database. The bias of the truncated estimator (6) is small for θ close to θ_{i+j} for all i, j, if only g is regular in a certain sense. Properties of estimator, bounded variance and asymptotic unbiaseness, (6) are analyzed in [12].

4 Experimental Study

We are interested in applications of the MDP framework to learning reactive policies of machines. In this section we analyze a challenging problem of this type, namely learning to run an emulated planar model of a large cat. The cat robot, called Half-Cheetah [11], is presented in Fig. 1. It is a planar kinematic string of 9 links, 8 joints, and 2 "paws". Because 5-th joint is fixed at 180^o, and its adjacent links have the same length, joints 4-th and 6-th are always at the same position; therefore, the object does not look like a string. The angles of 4-th and 5-th joint are fixed, all the the others are controllable. Consequently, Half-Cheetah is a model of a 6-degree-of-freedom walking robot.

In all the experiments the controlled system is emulated i.e., simulated in real time. A quantum of system's real time is equal to a quantum of the corresponding computer time, which means that the computer has a lot of spare time that can be devoted to parallel computations. The setup of our experiments is designed to closely resemble a situation when control of a physical machine is to be optimized in real time by means of learning.

The torque τ_i applied at i-th joint is calculated as

$$\tau_i = T_i \min\left\{\max\{-1, \tau_i^0 + a_i^0\}, 1\right\}$$

where τ_i^0 is a "spontaneous" torque at i-th joint, a_i^0 is the output of the learning controller, and T_i expresses "strength" of i-th joint. The spontaneous torque τ_i^0 is implemented as a PD-controller with saturation. It roughly stabilizes the i-th joint at its initial angle. We follow a typical setting of control system design: While it is usually relatively easy to provide a controller that stabilizes a system around a certain state (e.g. by using PD-controllers), it is much more difficult to design a controller that makes the system perform a certain nontrivial activity. In our paper, the controller is to *learn* to make Half-Cheetah run.

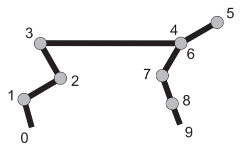

Fig. 1. The initial position of Half-Cheetah. It consists of 9 links, 8 joints among which 2 are fixed (4, 5), and 2 paws (0, 9)

An interested reader may find a detailed description of Half-Cheetah in [11]. The discussion here will be limited to the main facts: Half-Cheetah is about 1 meter long, weighs 10kg, and the ,,strengths" of its joints vary from 30 to 120Nm. It is designed to be a realistic model (as far as a planar model can be realistic) of a large, yet light cat.

In order to apply the reinforcement learning to Half-Cheetah, we must define the states and the rewards the learning algorithm has access to. Both are described in detail in [11]. Here we only mention that state is 31-dimensional and the main part of reward is speed measured in meters per second. There are also other parts that play various roles in early stages of learning: (i) a penalty for an attempt to apply torque from outside of the permissible interval, (ii) a penalty for the internal force that keeps the joint angle within its bounds (if i-th joint angle is equal to either of its bounds, then i-th ,,tendon" hurts the cat), (iii) a penalty for not moving the trunk up and keeping the paws on the ground when the animal is not moving forward, (iv) penalties for touching the ground with the heel, the knee, and the head.

The learning algorithm. In order to make Half-Cheetah run, we combine the Basic Actor-Critic and the idea of experience replay. The algorithm we apply (Replaying BAC, in short) is specified below.

Algorithm 3. The Basic Actor-Critic with Experience Replay (Replaying BAC).

```
0: $t := 1$. Initialize $\theta$ and $v$.
1: Draw and execute an action, $a_t \sim \pi(\cdot\,; s_t, \theta)$.
2: Register the tuple $\langle s_t, a_t, \theta, r_t, s_{t+1}\rangle$ in the database.
3: Make sure only $N$ most recent tuples remain in the database.
4: Repeat $\nu(t) = \min\{c_0, c_1 t\}$ times:
5:    Draw $i \in \{t-N+1, t-N+2, \dots, t\}$.
      Draw $K \sim Geom(\rho)$.
      Calculate $SUM$ equal to
      $\sum_{k=0}^{K} \alpha^k d_{i+k} \min\left\{\prod_{j=0}^{k} \frac{\pi(a_{i+j}; s_{i+j}, \theta)}{\pi(a_{i+j}; s_{i+j}, \theta_{i+j})}, b\right\}$
      for $d_{i+k} = r_{i+k} + \gamma \bar{V}(s_{i+k+1}; v) - \bar{V}(s_{i+k}; v)$.
6:    Adjust $\theta$ along an estimator of $\phi(s_i, \theta, v)$:
7:       $\theta := \theta + \beta_t^{\theta} \nabla_{\theta} \ln \pi(a_i; s_i, \theta) SUM$.
8:    Adjust $v$ along an estimator of $\psi(s_i, \theta, v)$:
9:       $v := v + \beta_t^{v} \nabla_{v} \bar{V}(s_i; v) SUM$.
10: Assign $t := t + 1$ and repeat from Step 1.
```

The policy applied to Half-Cheetah is comprised of two parts: a neural network and a normal distribution. The input of the network is the state. The output becomes a mean value of the normal distribution with covariance matrix $C = 5^2 I$. The distribution generates actions. The elements of the 6-dimensional action a are transformed into the control stimuli a^0 as $a^0_i = a_j/30$, where the indexes $i = 1, 2, 3, 6, 7, 8$ correspond to $j = 1, 2, 3, 4, 5, 6$, respectively.

The second approximator used by the learning algorithm is the critic, $\bar{V}$, i.e. the neural approximation of the value function. Both the critic network and the actor network have the form of two layer perceptron with linear output layer. Their hidden layers consist of M^A (the actor) and M^C (the critic) sigmoidal (arctan) elements. Each neuron has a constant input (bias). The initial weights of the hidden layers are drawn randomly from the normal distribution $N(0,1)$ and the initial weights of the output layers are set to zero.

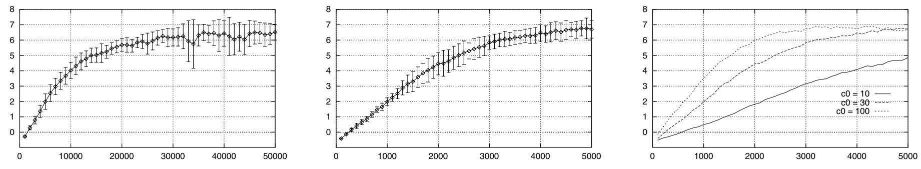

Fig. 2. Actor-Critics for Half-Cheetah: The average reward vs. trial number. *Left:* The Basic Actor-Critic. Each point averages 1000 consecutive trials. The curve averages 10 runs. The one-sigma limits are calculated to assess run-to-run variability of trial averages. *Middle:* The Basic AC with Experience Replay (Replaying BAC). Note that the number of trials in this figure is about 10 times smaller than that of the top figure for the basic method. The curve averages 5 runs and each point averages only 100 consecutive trials. *Right:* The Replaying BAC for various replaying intensity, c_0. Each curve averages 5 runs.

The parameters of the Basic Actor-Critic are as follows: (the actor) $M^A = 80$, $C = 5^2 I$, (the critic) $M^C = 160$, (step-sizes) $\beta_t^\theta \equiv \beta_t^v \equiv 5.10^{-5}$, (estimation) $\gamma = 0.99$, $\lambda = 0.9$. The resulting learning curves are depicted on the left-hand part of Fig. 2. The parameters of the replaying BAC are as follows: (the actor) $M^A = 80$, $C = 5^2 I$, (the critic) $M^C = 160$, (step-sizes) $\beta_t^\theta \equiv \beta_t^v \equiv 2.10^{-5}$, (database) $N = 3.10^4$, (estimation) $\gamma = \alpha = 0.99$, $\lambda = \rho = 0.9$, (computational effort) $c_0 = 30$, $c_1 = 0.3$. With a computer equipped with Intel Quad$^{\text{TM}}$Q9300, the simulations were carried on in real time of Half-Cheetah.

Experiments. Learning curves for the setting discussed above are shown in Fig. 2. A single trial lasts, on the average, for 5 sec. The left-hand part of Fig. 2 reports experiments with the Basic Actor-Critic applied to Half-Cheetah. It is seen, that the algorithm learns to control Half-Cheetah in about 3000 trials, which is about 42 hours of Half-Cheetah. The middle, and the right-hand part of Fig. 2 presents the averaged learning curve for the Replaying BAC applied to the same problem. The curve reports about 7 hours of learning. The algorithm learns to control Half-Cheetah in about 4500 trials, which is about 6 hours of Half-Cheetah time. It is interesting to observe the Half-Cheetah learned policy at various stages of learning (Fig. 3).

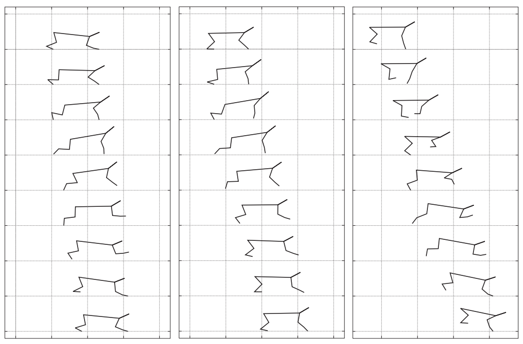

Fig. 3. Typical sequences of Half-Cheetah states at various stages of learning by the Basic Actor-Critic with Experience Replay (Replaying BAC).
Left: Awkward walk after 1 hour of training.
Middle: Trot after 2.5 hours of training.
Right: Nimble run after 7 hours of training.

Let us now analyze whether the concepts introduced in the paper indeed improve the quality and the speed of learning. The right part of Fig. 2 demonstrates how the intensity of the computation process translates into the speed of learning. It is seen that the larger intensity, the faster convergence. Plausibly for a certain large c_0, further increase of this parameter does not yield learning speed improvement. However, it is quite time-consuming to investigate high values of c_0. In fact, for $n > 30$ the computations are too slow to take place in real time of Half-Cheetah. The computer time of a single run is then proportional to c_0 and for $c_0 = 100$ it is around 31 hours. Obviously, it is only a matter of computer power. With a fast enough computer, the processing for $c_0 = 100$ could be performed in real time of Half-Cheetah. A satisfying policy could be then obtained after 3200 trails, which is about 4 hours of Half-Cheetah time.

5 Conclusions

In this paper we combined the technique of experience replay with a sequential Actor-Critic algorithm. Algorithms of this type deserve serious attention since they represent the most successful approach to applying reinforcement learning to realistic control tasks with continuous state and action spaces. As it has been verified experimentally, experience replay gives a radical learning speedup. The required number of interactions with the environment, which is critical for the applicability of reinforcement learning to real-world tasks, can be considerably reduced. For the fairly difficult Half-Cheetah task we observed a speedup factor of 10, allowing a satisfactory policy to be reached after as little as 4 hours of Half-Cheetah time (assuming availability of very large computation power), compared to about 42 hours required by the Basic Actor-Critic.

References

1. Bartlett, P.L., Baxter, J.: Stochastic optimization of controlled partially observable markov decision processes. In: Proc. of the 39th IEEE Conf. on Decision and Control (CDC 2000), vol. 1, pp. 124–129 (2000)
2. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can learn difficult learning control problems. IEEE Trans. on SMC 13, 834–846 (1983)
3. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Incremental natural actor-critic algorithms. In: Advances in NIPS, vol. 21 (2008)
4. Cichosz, P.: An analysis of experience replay in temporal difference learning. Cybernetics and Systems 30, 341–363 (1999)
5. Kimura, H., Kobayashi, S.: An analysis of actor/critic algorithm using eligibility traces: Reinforcement learning with imperfect value functions. In: Proc. of the 15th ICML, pp. 278–286 (1998)
6. Konda, V., Tsitsiklis, J.: Actor-critic algorithms. SIAM Journal on Control and Optimization 42(4), 1143–1166 (2003)
7. Lin, L.-J.: Reinforcement learning for robots using neural networks. Ph.D thesis, Carnegie Mellon University, Pittsburgh, PA, USA (1992)

8. Mahadevan, S., Connell, J.: Automatic programming of behavior-based robots using reinforcement learning. Artificial Intelligence 55(2-3), 311–365 (1992)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
10. Watkins, C., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
11. Wawrzyński, P.: Learning to control a 6-degree-of-freedom walking robot. In: Proc. of EUROCON 2007, pp. 698–705 (2007)
12. Wawrzyński, P., Pacut, A.: Truncated importance sampling for reinforcement learning with experience replay. In: Proc. CSIT Int. Multiconf., pp. 305–315 (2007)