Proceedings Paper:
Altahhan, A orcid.org/0000-0003-1133-7744 (2018) TD(0)-Replay: An Efficient Model-Free Planning with full Replay. In: 2018 International Joint Conference on Neural Networks (IJCNN). 2018 International Joint Conference on Neural Networks (IJCNN), 08-13 Jul 2018, Rio de Janeiro, Brazil. IEEE . ISBN 978-1-5090-6015-3

https://doi.org/10.1109/ijcnn.2018.8489300

TD(0)-Replay: An Efficient Model-Free Planning with full Replay

Abdulrahman Altahhan
School of Computing and Creative Technology
Leeds Beckett University
Leeds, UK.
a.altahhan@leedsbeckett.ac.uk

***Abstract*— In this paper we present a novel reinforcement learning method that allows for full replay of all past experience in every step of a reinforcement learning agent life with reasonable overhead. In particular, we show how to deduce an efficient equivalent backward view by replaying the full past experience online using TD(0) error for a linear model. We call the resultant methods TD(0)-Replay and Sarsa(0)-Replan, respectively. We emphasise the already established link between replaying and planning in our algorithm design by comparing it with an Extensive Dyna Planning algorithm, where we show that our method can outperform this expensive form of planning methods. We test the new methods on two different domain problems; Random Walk to test TD(0)-Replay prediction capabilities and Dyna Maze to test Sarsa(0)-Replan planning capabilities where we show that our algorithms dominate other replaying and planning methods.**

Keywords—TD(0)-Replay, Sarsa(0)-Replan, Dyna Planning, Efficient Planning, Full Replay, Planning by Replaying.

I. INTRODUCTION

In Reinforcement Learning, replaying past experience has been shown to have an important and definite role in reaching an optimal or close-to-optimal policy. Replay becomes even more important when dealing with experience that is difficult or expensive to simulate or when the trajectory of available experience is very limited. Especially in applications that requires real world interaction, it is difficult, and simply undesirable, to have to repeated the experience physically. Instead replaying the experience in the mind of the agent and learning form it becomes a natural and important method of learning. In essence, such complex task that needs rich imagination is going to be inevitably computationally expensive. Yet, in online learning, allowing the agent to *fully* repeat *all* past experience that has been done *so far* is even more demanding. However, this full repetition if achieved efficiently, can provide the agent with a very powerful learning mechanism that boosts its performance and allows it to maximise the so far lived experience in a way that has not been done before.

In this paper we will provide a new method that allows the agent to achieve exactly the above. I.e. to allow the agent to fully replay all past experience (in its head rather that physically), which will allow it, in turn, to quickly optimise its value function prediction as well as improve its policy. We will show that this can be achieved with a reasonable computational expense that makes the proposed method, called TD-Replay, a very attractive method for the above mentioned situations which requires maximisation of so far experience without having to physically repeat it. [1] and [2] for example studied the effect of replaying and they have shown that the agent can boost its experience when using replaying. The original interpretation for experience replay is that the samples will be presented for the agent as a new set of samples [1]. While in [3] the agent suffices by re-updating its weight as well as its value function estimation either fully or partially. At the same time, [4] studied the effect of replaying from planning perspective and they showed that replaying can be looked at as planning by looking into the past (instead of future) and they showed that their TD(0) replay algorithm is equivalent to the Planning with the linear Dyna model Algorithm. We will follow a similar approach as in [4] and [5], however we will develop a different algorithm that has its own update rules and mechanism that is different form the presented algorithms but has some resemblance in terms of the form of updates.

In our work we will introduce an efficient algorithm with forward view that depends on the mechanism provided by [4] and [5]. True Online TD [5] allows the learning process to be repeated for all $k = 1 \dots t$ but it does not utilise replay, it assumes that the agent will always reinitialise its weights to the same initial values in every time step t. On the other hand, [4] utilises replaying on the level of targets only. Replaying TD(0) updates Algorithm for example (and its more efficient equivalent Planning with Dyna Algorithm) assumes that the agent starts from the same initial weights at every set of imaginary experience, only its U_i targets are changing according to the latest weights coming from the real time step t. Our algorithm, which we call TD(0)-Replay, assumes that both the targets as well as the initial weights are changing in every set of imaginary updates $k = 1 \dots t$, making it more vigilant and adaptive to changes in the environment.

TD(0)-Replay will depends on the usual TD error and will not reinitialise the weights in each time step t, instead it will assume that the starting weights of time step t are those obtained after updating the weights in time step $t-1$, but will assume that the agent is going to replay all of its past experience for each imaginary time step k, where $k = 1 \dots t$.

II. TD-REPLAY FORWARD VIEW

A. TD(0)-Replay update rules at time step t

In this section we will develop the updates rules for our TD-Replay method on the basis of a one-layer neural network model (linear model). At time step t TD(0) update for a linear model is given as

$$\delta_t = R_{t+1} + \gamma\theta_t^\intercal\phi_{t+1} - \theta_t^\intercal\phi_t \quad (1)$$

$$\theta_{t+1} = \theta_t + \alpha_t\delta_t\phi_t \quad (2)$$

Where: δ_t is the temporal Difference error, R_{t+1} is the reward signal, γ is a discount factor, $\phi_t^\intercal$ is the transpose of feature vector ϕ_t obtained through current state S_t, $\theta_t^\intercal$ is the transpose of the weight vector θ_t and α_t is a learning step; all varies according to time step t.

In order to replay previous experience and assuming that the agent is at time step t, in this case the TD(0) error for a time step k: $0 \le k < t$ is given by

$$\theta_{k+1} = \theta_k + \alpha_k(R_{k+1} + \gamma\theta_k^\intercal\phi_{k+1} - \theta_k^\intercal\phi_k)\phi_k \quad (3)$$

Or

$$\theta_{k+1} = [I + \alpha_k\phi_k(\gamma\phi_{k+1}^\intercal - \phi_k^\intercal)]\theta_k + \alpha_k R_{k+1}\phi_k \quad (4)$$

Hence by renaming we have:

$$\theta_{k+1} = A_k\theta_k + B_k \quad (5)$$

$$A_k := [I_{n\times n} + \alpha_k\phi_k(\gamma\phi_{k+1} - \phi_k)^\intercal] \quad (6)$$

$$B_k := \alpha_k\phi_k R_{k+1} \quad (7)$$

Where n is the features dimension i.e. $|\phi_k| = n$. Of course since we are using a linear model we have also $|\theta_k| = n$. A_k is an $n \times n$ squared matrix, while B_k is $n \times 1$ vector, $\intercal$ is the transpose symbol.

B. Cummulative update rules at time steps $t = k + 1$

If we allow the agent to replay all so far experience in every step, we get the following formulae:

$$\begin{array}{lll} t=1 & k=0 & \theta_1^1 = A_0\theta_{init} + B_0) \\ t=2 & k=0 & \theta_1^2 = A_0\theta_1^1 + B_0 \\ & k=1 & \theta_2^2 = A_1\theta_1^2 + B_1 \\ t=3 & k=0 & \theta_1^3 = A_0\theta_2^2 + B_0 \\ & k=1 & \theta_2^3 = A_1\theta_1^3 + B_1 \\ & k=2 & \theta_3^3 = A_2\theta_2^3 + B_2 \end{array} \quad (8)$$

…

By convention, since the last imaginary step is $k = t - 1$ for t, (and the last update is going to be $\theta_{k+1}^t = \theta_t^t$), we define $\theta_t := \theta_t^t$. The above will give the following set of formula for θ_t^t.

$$\theta_1 = A_0\theta_0^0 + B_0$$
$$\theta_2 = A_1A_0\theta_1 + A_1B_0 + B_1$$
$$\theta_3 = A_2A_1A_0\theta_2 + A_2A_1B_0 + A_2B_1 + B_2$$

…

$$\theta_{t+1} = A_t..A_0\theta_t + A_t..A_1B_0 + \cdots + A_tB_{t-1} + B_t \quad (9)$$

By defining

$$A_t^i := A_t \ldots A_i \quad (10)$$

$$A_t^{t+1} := I_{n\times n} \quad (11)$$

The previous equations can be written through induction as

$$\theta_{t+1} = A_t^0\theta_t + A_t^1B_0 + \cdots + A_t^tB_{t-1} + A_t^{t+1}B_t \quad (12)$$

Where: $A_t := [I_{n\times n} + \alpha_t\phi_t(\gamma\phi_{t+1} - \phi_t)^\intercal]$

$$\theta_{t+1} = A_t^0\theta_t + \sum_{i=0}^t A_t^{i+1}B_i \quad (13)$$

Hence by defining the eligibility trace e_t and eligibility matrix $\hat{e}_t$ as:

$$\hat{e}_t := A_t^0 \quad (14)$$

$$e_t := \sum_{i=0}^t A_t^{i+1}B_i \quad (15)$$

The updates rules for the learning weights can be written as:

$$\theta_{t+1} = \hat{e}_t\theta_t + e_t \quad (16)$$

C. Incremental cummulative update rules at time steps $t = k + 1$

Let us now deduce incremental rules for the eligibility trace e_t and eligibility matrix $\hat{e}_t$.

As for the eligibility matrix $\hat{e}_t$, by induction we have:

$$\hat{e}_t = A_t^0 = A_tA_{t-1} \ldots A_0 = A_tA_{t-1}^0 = A_t\hat{e}_{t-1}.$$

While for the eligibility trace e_t, we have

$e_t = \sum_{i=0}^t A_t^{i+1}B_i = A_t\sum_{i=0}^{t-1} A_{t-1}^{i+1}B_i + B_t = A_te_{t-1} + B_t$, since we have $e_{t-1} = \sum_{i=0}^{t-1} A_{t-1}^{i+1}B_i$.

Hence our TD(0)-Replay algorithm can be written in the following order:

$$A_t = [I_{n\times n} + \alpha_t\phi_t(\gamma\phi_{t+1} - \phi_t)^\intercal] \quad (17)$$

$$B_t = \alpha_t\phi_t R_{t+1} \quad (18)$$

$$e_t = A_te_{t-1} + B_t \quad (19)$$

$$\hat{e}_t = A_t\hat{e}_{t-1} \quad (20)$$

$$\theta_{t+1} = \hat{e}_t\theta_t + e_t \quad (21)$$

where we have $A_0 = \hat{e}_0 = I_{n\times n}, B_0 = \alpha_0\phi_0R_1, e_0 = 0_{n\times 1}$. It should be noted that the algorithm uses just current time step information to apply a full replay of all past experience hence its significance lies in this particular characteristic.

D. Efficient Form of TD(0)-Replay Forward

TD-Replay in its previous form can be made more efficient by unpacking A_t in the updates and replacing matrix multiplications in (19) and (20) with matrix to vector multiplication.

As for the calculations of e_t we have $e_t = A_t e_{t-1} + B_t$, hence $e_t = e_{t-1} + \alpha_t \phi_t [(\gamma \phi_{t+1} - \phi_t)^\intercal e_{t-1}] + B_t$. Therefore the equation involves calculating the term $(\gamma \phi_{t+1} - \phi_t)^\intercal e_{t-1}$ which is a scalar (multiplying a vector e_{t-1} by a vector transpose) and it is more efficient than multiplying a squared matrix A_t and a vector e_{t-1}.

As for $\hat{e}_t$ we have $\hat{e}_t = A_t \hat{e}_{t-1}$, hence $\hat{e}_t = \hat{e}_{t-1} + \alpha_t \phi_t [(\gamma \phi_{t+1} - \phi_t)^\intercal \hat{e}_{t-1}]$. It should be noted that $[(\gamma \phi_{t+1} - \phi_t)^\intercal \hat{e}_{t-1}]$ is a multiplication of a squared matrix $\hat{e}_{t-1}$ by a vector $(\gamma \phi_{t+1} - \phi_t)^\intercal$ and is more efficient than multiplying two squared matrices A_t and $\hat{e}_{t-1}$ as in $A_t \hat{e}_{t-1}$. The complexity still lies within this calculation which is of $O(n^2)$ for space and time. The final algorithm can be written as:

$$e_t = e_{t-1} + \alpha_t \phi_t [(\gamma \phi_{t+1} - \phi_t)^\intercal e_{t-1} + R_{t+1}] \quad (22)$$

$$\hat{e}_t = \hat{e}_{t-1} + \alpha_t \phi_t [(\gamma \phi_{t+1} - \phi_t)^\intercal \hat{e}_{t-1}] \quad (23)$$

$$\theta_{t+1} = \hat{e}_t \theta_t + e_t \quad (24)$$

Where we have $e_0 = 0_{n\times 1}$, $\hat{e}_0 = I_{n\times n}$

Surely the complexity can be reduced by using binary features or some other sparsity reduction techniques if the features are binary. In those case if the number of present features is m where m<n then the complexity becomes $O(n \times m)$.

E. Comparing TD(0)-Replay with Other Replay Algorithms

Clearly from a formative perspective the updates rules are different, in form, than those presented in [4]. From a fundamental inner working mechanism perspective, the main differences between 'TD(0)-Replay' and 'Replaying TD(0)' Algorithms [4] (and to some extent even the True online TD algorithms [5]) are in the following essential characteristics. The first, is that the mechanics are different since in [4] the agent uses old weights that resulted from previous steps $1 \dots t$ in calculating the targets U_k (in all of the imaginary steps $k = 0 \dots t$). While, TD(0)-Replay uses the weights θ_t^t from latest experience t as an initial weights in step $k = 0$, then it lets the consequent updates specify targets U_k that has been recalculated using the latest θ_k^{t+1} in consequent steps $k = 1 \dots t$. Secondly, the term A_t is different since we have $A_t = [I_{n\times n} + \alpha_t \phi_t (\gamma \phi_{t+1} - \phi_t)^\intercal]$ instead of $A_t = [I_{n\times n} - \alpha_t \phi_t (\phi_t)^\intercal]$. Thirdly, the broader concept of weight updates with no reinitialisation makes our approach more general. In other words for this form of TD-Replay we do not reinitialise the weights in the start of each imaginary set of updates to a fixed weights θ_{init}, unlike [4], [5] and [10]. Later we will relax this assumption to obtain a new way to write TD(0) algorithm (involving eligibility trace) making TD(0) a special case of TD(0)-Replay. Finally, our form of replaying is in the traditional sense where old samples are introduced as new samples [11] as opposed to re-evaluating done by Replaying TD(0), where the agent is rather than introducing old samples as new samples, it re-revaluate its target approximation and redo its update for a number of times or until convergence.

On the other hand, both TD(0)-Replay and Replaying TD(0), replay (albeit in a different sense) fully all previous experience in every steps t. Also both do not require storing any previous states or weights, hence they are efficient. TD-Replay requires storing the eligibility squared matrix $\hat{e}$ and vector e (equivalent requirements for Replaying TD(0) algorithm updates is to store square matrix F and vector b). The complexities of both our algorithm and their algorithm for storage and computation in each time step are of $O(n^2)$ in the worst case regardless of the number of steps, however Replaying TD(0) cost k times more computation resources than TD(0)-Replay due to the loop of target update refinement.

Formula (22) -(24) set of update rules define a replaying algorithm for an agent to be able to predict the value function for a specific task in some environment. The algorithm is given below.

Algorithm 1 *TD(0)-Replay: TD(0) Planning by Replaying Full Past Experience, Value Function Prediction*

INPUT: $\alpha, \gamma, \theta_{init}$
$\theta \leftarrow \theta_{init}$
Loop (over episodes):
  Obtain initial S, ϕ
  $\hat{\boldsymbol{e}} \leftarrow I_{n\times n}$, $\boldsymbol{e} \leftarrow \mathbf{0}_{n\times 1}$
  While (terminal state has not been reached), do:
    act according to the policy
    observe next reward $R = R_{t+1}$, next state $\acute{S} = S_{t+1}$
    and its features $\acute{\phi} = \phi_{t+1}$
    $\alpha \leftarrow \ell(\alpha)$
    $\boldsymbol{e} \leftarrow \boldsymbol{e} + \alpha \boldsymbol{\phi} \left((\gamma \acute{\boldsymbol{\phi}} - \boldsymbol{\phi})^\intercal \boldsymbol{e} + R \right)$
    $\hat{\boldsymbol{e}} \leftarrow \hat{\boldsymbol{e}} + \alpha \boldsymbol{\phi} \left[(\gamma \acute{\boldsymbol{\phi}} - \boldsymbol{\phi})^\intercal \hat{\boldsymbol{e}} \right]$ (replaying)
    $\boldsymbol{\theta} \leftarrow \hat{\boldsymbol{e}} \boldsymbol{\theta} + \boldsymbol{e}$
    $\boldsymbol{\phi} \leftarrow \acute{\boldsymbol{\phi}}$

It should be noted that the agent is replaying all past experience and updates the weights accordingly. In the above algorithm the learning rate α is assigned at every time steps t according to $\ell(\alpha)$ which can be any scheme that reduces α (annealing for example). In practices if α is chosen to be small enough then it can be left without updating it [3].

F. TD(0)≡TD(0)-Replay with Reinitialisation

If we changed the mechanism of TD(0)-Replay so that it will reinitialise its weights in each imaginary step $k = 0$, to $\theta_0^t = \theta_{init}\ \forall t$ then the algorithm goes back into a TD(0) one-go update rule that summarises in each step what has been done in all past steps, this is useful since it will allow exact equivalent between online and offline updates, but also it may allow us to boost the performance through accumulation although we will leave this for future work.

When initialising the weights in each step to θ_{init}, then the update rules (17)-(20) stay the same, while update rule (21) becomes

$$\theta_{t+1} = \hat{e}_t \theta_{init} + e_t \quad (25)$$

and the next update is going to be

$$\theta_{t+2} = \hat{e}_{t+1} \theta_{init} + e_{t+1}$$

In this case, since θ_{init} is fixed then we can make the updates rules much more efficient since we can store $\hat{e}_t \theta_{init}$ instead of $\hat{e}_t$. Hence, the update rule (20) can be changed into:

$$\hat{e}_t \theta_{init} = A_t \hat{e}_{t-1} \theta_{init} \quad (26)$$

By defining $\acute{\theta}_t$ as a vector

$$\acute{\theta}_t := \hat{e}_t \theta_{init} \quad (27)$$

The update (28) can be written as:

$$\acute{\theta}_t = A_t \acute{\theta}_{t-1} \quad (28)$$

Hence, these updates along with (22) define a new algorithm that can be written efficiently as

$$e_t = e_{t-1} + \alpha_t \phi_t [(\gamma \phi_{t+1} - \phi_t)^\intercal e_{t-1} + R_{t+1}] \quad (29)$$

$$\acute{\theta}_t = \acute{\theta}_{t-1} + \alpha_t \phi_t \left[(\gamma \phi_{t+1} - \phi_t)^\intercal \acute{\theta}_{t-1}\right] \quad (30)$$

$$\theta_{t+1} = \acute{\theta}_t + e_t \quad (31)$$

Where we have $e_0 = 0_{n\times 1}$, $\acute{\theta}_0 = \theta_{init}$ all as vectors. The algorithm is shown below. One important issue to realise in this algorithm is the initialisation to $\acute{\theta} \leftarrow \theta$ in each episode. This is important so that the updates of θ are not lost. This new way of writing *TD(0)* algorithm is doing a special type of efficient replay that resemble the one done in True Online TD but it uses one step backup rather than the full interim λ-returns.

Algorithm 2 *TD(0): TD(0)-Replay(0), Value Function Prediction(Policy Evaluation)*

INPUT: $\alpha, \gamma, \theta_{init}$
$\theta \leftarrow \theta_{init}$
Loop (over episodes):
  Obtain initial S, ϕ
  $\acute{\boldsymbol{\theta}} \leftarrow \theta$, $\boldsymbol{e} \leftarrow \mathbf{0}_{n\times 1}$
  While (terminal state has not been reached), do:
    act according to the policy
    observe next reward $R = R_{t+1}$, next state $\acute{S} = S_{t+1}$
    and its features $\acute{\phi} = \phi_{t+1}$
    $\alpha \leftarrow \ell(\alpha)$
    $\boldsymbol{e} \leftarrow \boldsymbol{e} + \alpha \boldsymbol{\phi} \left(\left(\gamma \acute{\boldsymbol{\phi}} - \boldsymbol{\phi}\right)^\intercal \boldsymbol{e} + R \right)$
    $\acute{\boldsymbol{\theta}} \leftarrow \acute{\boldsymbol{\theta}} + \alpha \boldsymbol{\phi} \left(\left(\gamma \acute{\boldsymbol{\phi}} - \boldsymbol{\phi}\right)^\intercal \acute{\boldsymbol{\theta}} \right)$
    $\boldsymbol{\theta} \leftarrow \acute{\boldsymbol{\theta}} + \boldsymbol{e}$
    $\boldsymbol{\phi} \leftarrow \acute{\boldsymbol{\phi}}$

Algorithm 1 has exact equivalent algorithm that is based on some form of repetitive TD updates. *TD(0)-Replay* can also be generalised by using the full interim λ-returns but this will be left for future work. We will call Algorithm 2 *TD(0)-Replay(0)* to recognise the fact that in each step it starts from the initial weight θ_0 which renders *TD(0)-Replay* to have no replaying, but the algorithm becomes computationally cheaper since as we said before it is equivalent to TD(0). These ideas are useful to parametrise between full replay, TD(0)-Replay(1), and no replay, TD(0)-Replay(0), the parametrisation will be left for future work.

G. TD(0)-Replay and Re-Planning

From control perspective, since we are updating the weights then reflecting back on the agent past experience to learn from it, it is only reasonable to consider our algorithm as a re-planning algorithm as well. $\hat{e}$ can be considered to be trying to establish a prediction model for the feature vector that is capable of being changed according to the difference between the two consequent vectors $\left(\gamma \acute{\boldsymbol{\phi}} - \boldsymbol{\phi}\right)^\intercal$ instead of predicting the next feature vector as in [3]. Similarly, TD(0)-Replay needs to know how the reward function will change, through e , in order to come up with a planning scheme for the future replays of past experience according to the latest up-to-date weights and targets. It should be noted that TD(0)-Replay is different than an algorithm that utilises the residual gradient since the term $\left(\gamma \acute{\phi} - \phi\right)^\intercal \hat{e}$ is multiplied by $\alpha\phi$ not with $\left(\gamma \acute{\phi} - \phi\right)^\intercal$.

H. Sarsa(0)-Replay Forward Algorithm

As for control, we can straightforwardly build control algorithm on the grounds of TD-Replay. The agent would need to learn a suitable policy; the policy can be deduced out of the predicted value function for the agent.

One way to build the policy learning model, based on the value prediction model, is as follows. A set of learning weights is provided for each action, given the set of actions are limited. The control features ψ_t are going to have a cardinality of $|\psi_t| = |\mathcal{A}| \times n$ where $n = |\phi_t|$ (the state feature size) and $|\mathcal{A}|$ is the number of actions that an agent can take at any time step t. The agent also would have the same size for its weights when using a linear model i.e. $|\mathcal{A}| \times n$ weights.

In each time step, the set of features $\phi_t(a_t)$ in ψ_t (that corresponds to the current action a_t) will be populated with the values of the state features, while the rest $\phi_t(a_i)$: $a_i \neq a_t$ in ψ_t will be simply populated with 0. Hence the learning takes place on the set of weights corresponding to the current actions since the rest of the features are going to be 0.

In order to deduce a suitable policy, the agent calculates the value function for each action and then picks the action with the highest value (most of the time, except for few times with small probability of ε where the agent picks a random action). This type of policy, called ε-greedy policy, is a common policy to be followed, other policies such as soft-max is also possible.

In this paper we will follow an ε-greedy policy. By doing the above scheme a similar algorithm can be written for the agent in order to learn a suitable policy instead of learning only to predict the value function of its current policy. According to the policy improvement theorem this scheme of improving the policy by picking the max action value then updating the prediction accordingly will lead to convergence to an optimal policy in the case of a linear model that is being updated

according to the TD error, with some extra conditions on the learning rate [6][7] . The policy improvement algorithm for *Sarsa(0)-Replan* is given below. It should be noted that by convention $\acute{\psi} \leftarrow 0$ if $\acute{S}$ is terminal.

Algorithm 3 *Sarsa(0)-Replan: Policy Improvement*

INPUT: $\alpha, \gamma, \theta_{init}$
$\theta \leftarrow \theta_{init}$
Loop (over episodes):
  Obtain initial S, ϕ
  Select action A based on State S
  $\psi \leftarrow$ features corresponding to S, A; $(|\psi| = n \times |\mathcal{A}| = \mathcal{N})$
  $\hat{\boldsymbol{e}} \leftarrow I_{\mathcal{N}\times\mathcal{N}}$, $\boldsymbol{e} \leftarrow \boldsymbol{\theta} \leftarrow \mathbf{0}_{\mathcal{N}\times 1}$
  While terminal state has not been reached, do:
    take action A, observer next state $\acute{S}$ and reward R
    $a \leftarrow \varepsilon$-greedy(argmax$\acute{Q}(A_i) \leftarrow \boldsymbol{\theta}^{\intercal}(\boldsymbol{a_i})\acute{\boldsymbol{\phi}}$) $(|\acute{Q}| = |\mathcal{A}|)$
    $\acute{\psi} \leftarrow$ features corresponding to $\acute{S}, \acute{a}$
    $\alpha \leftarrow l(\alpha)$
    $\boldsymbol{e} \leftarrow \boldsymbol{e} + \alpha\boldsymbol{\psi}\left(\left(\gamma\acute{\boldsymbol{\psi}} - \boldsymbol{\psi}\right)^{\intercal}\boldsymbol{e} + R\right)$
    $\hat{\boldsymbol{e}} \leftarrow \hat{\boldsymbol{e}} + \alpha\boldsymbol{\psi}\left[\left(\gamma\acute{\boldsymbol{\psi}} - \boldsymbol{\psi}\right)^{\intercal}\hat{\boldsymbol{e}}\right]$ (planning by replaying)
    $\boldsymbol{\theta} \leftarrow \hat{\boldsymbol{e}}\boldsymbol{\theta} + \boldsymbol{e}$
    $\boldsymbol{\psi} \leftarrow \acute{\boldsymbol{\psi}}$; $A \leftarrow \acute{A}$

It should be noted that the agent effectively takes into consideration current step information (rewards and feature), updates the weights, replay all past experience and updates the weights according to the replay.

One issue should be mentioned here is that the algorithm is performing on-policy learning while replaying past experience using last step weight. Replaying is normally done off-policy because if the agent repeated the past experience without reinitialising its weights in each set of imaginary steps then it might choose different actions and end up going through different set of states. However, in practices as we shall see later since we are replaying the full past experience, this seems to balance out the bias in choosing specific samples and keep the agent in harmony with its current experience. From a learning perspective the weights get initialised at each time step into a different new starting point therefore by keeping the learning step relatively small we can still keep the on-policy learning process stable and avoid divergence. In future work we will tackle this issue and provide an off-policy version of *TD-Replay*. One important advantage of reinitialising to latest weights in each step is adaptation to a changing environment.

In order to improve the policy an agent can run either indefinitely or through a set of episodes that is specified in priori, where it stops when the learning slows down under a specific threshold. For our comparisons we will choose a specific number of episodes and compare the Route Mean Squared Error (RMSE) or the total number of steps to for those episodes.

I. Dyna Full Planning Algorithm

In order to compare our algorithms objectively we will compare *Sarsa(0)-Replan* with a special version of *Dyna Planning* which we call *Dyna Full Replanning*, where the agent regenerate (reimagine) fully all previous samples in every time step in order to better plan what to do with them in case it sees them in the future.

It seems reasonable that the concept of re-planning based on replaying past experience, works especially when the agent is expected to *revisit* some states due to its incompetence of reaching its terminal state or achieving its final goal (the case in lots of RL environment and tasks). The *Dyna Full Replanning* algorithm, shown below, is expensive because its complexity is $O(Tn^2)$ rather than $O(n^2)$, where n is the feature size and T is the total number of visited states. T is normally >> n especially at the start of learning. This extreme case of planning is conceived as the maximum performance a *Dyna Full Replan* algorithm can achieve. We will compare this algorithm performance with ours to show the real planning capabilities of *Sarsa-Replan* algorithms. Similar to the other algorithms, a policy improvement algorithm can be devised based on it.

Algorithm 4 *Dyna Full Replan: Extreme Planning (expensive; for comparison only)*

INPUT: $\alpha, \gamma, \theta_{init}$
$\theta \leftarrow \theta_{init}$
Loop (over episodes):
  Obtain initial S, ϕ
  $F \leftarrow \mathbf{0}_{n\times n}$, $\boldsymbol{e} \leftarrow \mathbf{0}_{n\times 1}$ $t \leftarrow 1$
  While (terminal state has not been reached), do:
    act according to the policy observe next reward R , $\acute{S}$ and $\acute{\phi}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\left[R + \boldsymbol{\theta}^{\intercal}\acute{\boldsymbol{\phi}} - \boldsymbol{\theta}^{\intercal}\boldsymbol{\phi}\right]\boldsymbol{\phi}$
    $\boldsymbol{F} \leftarrow \boldsymbol{F} + \alpha\left[\gamma\acute{\boldsymbol{\phi}} - \boldsymbol{F}\boldsymbol{\phi}\right]\boldsymbol{\phi}^{\intercal}$ (F is to predict next state $\acute{\boldsymbol{\phi}}$)
    $\boldsymbol{b} \leftarrow \boldsymbol{b} + \alpha(R - \boldsymbol{b}^{\intercal}\boldsymbol{\phi})\boldsymbol{\phi}$ (b is to predict next reward R)
    $\boldsymbol{\phi_t} \leftarrow \boldsymbol{\phi}$ (store visited state features, memory expensive)
    For $k \leftarrow 1$ *to* t (planning steps, computationally expensive)
      based on revisiting past states
      in order to obtain a better policy evaluation
      $\acute{\boldsymbol{\phi}} \leftarrow \boldsymbol{F}\boldsymbol{\phi_k}$
      $R \leftarrow \boldsymbol{b}^{\intercal}\boldsymbol{\phi_k}$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\left[R + \boldsymbol{\theta}^{\intercal}\acute{\boldsymbol{\phi}}_{\boldsymbol{k}} - \boldsymbol{\theta}^{\intercal}\boldsymbol{\phi_k}\right]\boldsymbol{\phi_k}$
    $t \leftarrow t + 1$

III. Experiments Design and Algorithms Testing

We have tested our algorithm on a two testbeds. The first is Random Walk which is a Markov Reward Process (MRP) to test the TD(0)-Replay prediction algorithm. MRPs are useful tools to isolate the prediction problem from the policy improvement (control) problem. The idea behind it is to assign an action, in each step, based on a transition probability that represents the dynamics of the environment. The actions are generated due to this probability only, there is no deliberate decision making taking place and the policy is stochastic with fix probabilities. Therefore, no policy improvement is taking place, only prediction improvement is happening in the underlining value function model. We will use a 6-state Random Walk environment, where the process starts form a middle state as in Fig. 1. The current state will be moved to the state in the left according to a probability of p, or to the right according to the probability 1-p. Once the process reaches the final state to the right the process stops and the agent is rewarded +1, otherwise if the process reaches the final state to the left, the process stops

and the agent is rewarded with 0. All other transitions have a 0 reward and hence the value function v(St) represents the probability of starting with state St and ending up in state E.

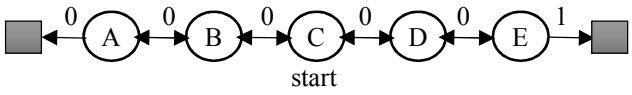

Fig. 1. *Random Walk for 6 states. The true value for those states are their probabilty of reaching the far left terminal state assuming the agent start form each of them.*

We have set $\gamma = 1$ and used a simple set of binary features that each represents a state. The features size is equal to the number of states. We have studies the effect of the learning rate for *TD(0)-Replay* in comparison with Replaying TD(0)=*Dyna Planning* algorithms. Fig. 2 shows the results. Although TD(0)-Replay may inquire more penalties but it favoured aggressive approach that allowed it to converge quickly to the required solution for the task in hand better than any other algorithm for most of the learning step values. This is ideal for expensive-experience application especially when less than a handful of episodes are available for the agent.

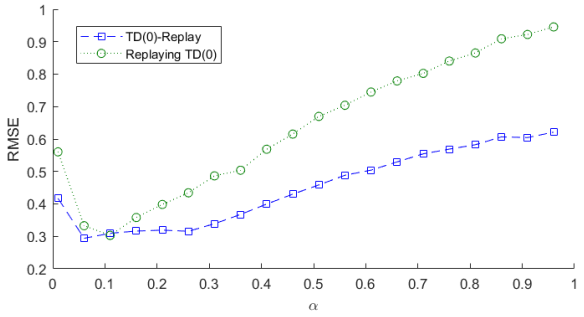

Fig. 2. *Comparison of RMSE of TD(0)-Replay and Replaying TD=Dyna Planning, algorithms for the Random Walk problem,T=500, 10 episodes and 1000 trials and different learning rates. Clearly TD(0)-Replay had a minimum RMSE and it performed best for α<0.1.*

The second test bed is the traditional Dyna Maze environment with 9 × 6 cells (states). The main goal of a Dyna Maze is for an agent to be able to learn a policy that allows it to reach a specific goal state, where it starts from a specific initial state (cell) in each episode. The episode ends when the agent reaches the goal square. Plenty of obstacles have been placed in the way from the Start state to the goal state (as can be seen in the dark grey squares in Fig.3). The agent is rewarded with -1 in each step wasted before reaching the goal while the reward for reaching the goal state is 0. An example of a simulated agent represented as a red square is shown in Fig. 3.

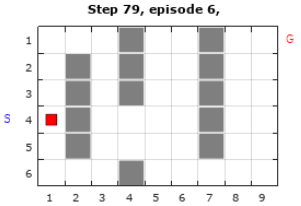

Fig. 3. *Dyna Maze example of the small environment*

Dyna Maze is traditionally used to test planning algorithms, we show here that *Sarsa(0)-Replan* exceeds the performance of other RL algorithms that involves planning. It actually competes head to head with very expensive replanning algorithm, *Dyna Full Replan*, that covers all previously visited states in each current state update. It should be noted that in the below experiments all results has been produced after averaging 20 runs. We have used binary features to represent the states by linearizing the domain. We have 54 states, each state is represented as a vector with 54 features, and we set $\gamma = .95$.

Fig. 4 shows the results for the *Dyna Full Replan* algorithm. It should be mentioned that this algorithm is included for comparison purpose only since it is expensive and can be impractical. Fig. 5 shows the comparison for *Sarsa(0)-Replan,* and for completeness of coverage Fig. 6 shows the comparisons results for different learning steps for *Sarsa(0)=Sarsa(0)-Replay(0)*.

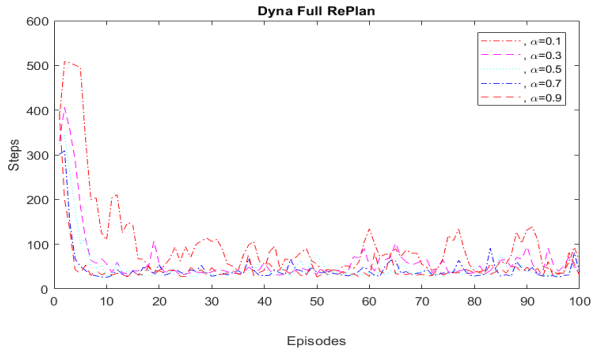

Fig. 4. *Compariosn of different learning steps for Dyna Full Replan for control averaged over 20 runs for binary features. Clearly 0.9 is best and the algorithm converges very quickly after the 5th episode in all cases, lower alpha gave more stability for the algorithm. But the algorithm is very slow and inefficent(Complexity is $O(Tn^2)$ in terms of time and memory, T is the total numberof steps) because the agent resamples all of the so far visited states in every time step.*

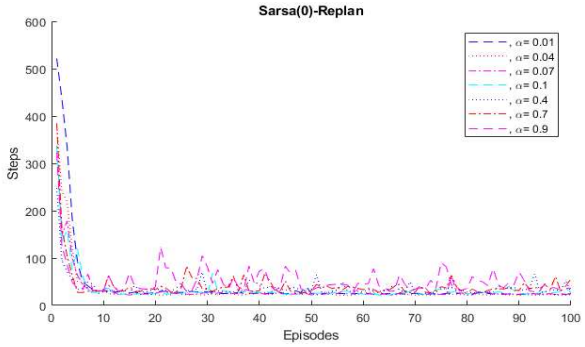

Fig. 5. *Compariosn of different learning steps for Sarsa(0)-Replan averaged over 20 runs for binary features.α < 0.5 gave best performance and the algorithm converges quickly after the 5th episode in most cases, lower α gave more stability for the algorithm. The algorithm is efficent in terms of complexity although the agent is effectively resamples all of the so far visited states in every time step.*

Fig. 7 shows a comparison of *Sarsa(0)-Replan* along with the other three algorithms. It is clear that our algorithm exceeds

the performance of other algorithms in this domain. It should be noted that *Sarsa(0)-Replan* has the advantage of being efficient with complexity of $O(n^2)$ for each time step calculation, has no hyper parameters to be set (such as λ), it is model free and effective for planning and it has an exact online equivalence algorithm that it was built on it; making it also truly online.

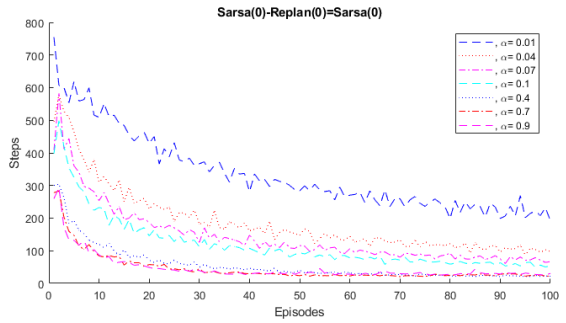

Fig. 6. *Compariosn of different learning steps for Sarsa(0)-Replan(0)= Sarsa(0) averaged over 20 runs for binary features.*

Fig. 8 shows the execution time for each algorithm. *Sarsa(0)-Replan* has a good execution time due to its efficiency in converging quickly to an optimal policy. It also can be seen that *Dyna Full Planning* is the most expensive as expected. Combining execution time along with how fast the algorithm reached the optimal policy, we conclude that *Sarsa(0)-Replan* can be the method of choice for those application that needs to maximise the lived experience, such physical and real time system.

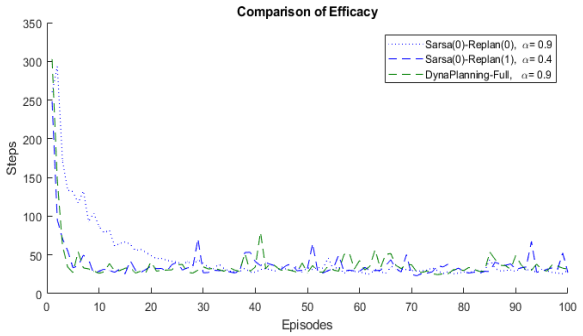

Fig. 7. *Final compariosn between Sarsa(0)-Replan(1)[complexity is $O(n \times m)$ m is the numebr of active features], Dyna Full Replanning for Control [complexity is $O(Tn^2)$ T is the number of episode steps], and Sarsa(0)-Replan(0)=Sarsa(0). Sarsa(0)-Replan(1) has done almost the same job as Dyna Full Replan with much better cost. This algorithm is an efficent model-free with reaosnable overhead and maximal performance. The learning rates have been chosen to maximise the perofrmance of each algorihtm as per the previous set of expeiments.*

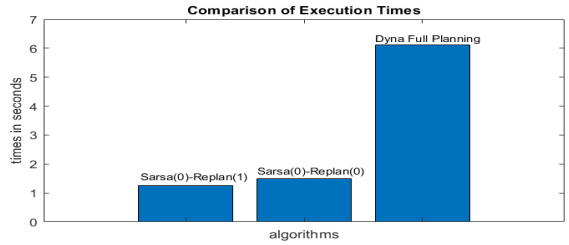

Fig. 8. *Compariosn different algorithms execution times.*

IV. Conclusion and Future Work

In this paper we have presented three new RL algorithms that allow for an efficient and full replay of all past experience in every step for a reinforcement learning agent life with reasonable overhead. TD(0)-Replay and Sarsa(0)-Replan seem to dominate other replaying and planning algorithms with similar complexity and has the advantage of being adaptive to environment changes. The presented algorithms are suitable for real time and experience-expensive systems (where learning through experience is a hard and expensive process). We showed how to deduce a backward view directly from the forward view for the online case. We have contrasted the presented algorithms with other similar algorithms and showed the differences in terms of mechanics and practically through experiments. Our experiments confirm the potential for TD-Replay method to be used in different domains and to exceed other replay and planning schemes.

In the future we will be looking towards generalising our algorithm to have a general target U_i instead of the one-step target. This new target will change the form of the matrix A and special attention would be needed to deduce an efficient form for the full replay of past experience. Another avenue is to develop the algorithm for a non-linear model, and also to develop the algorithm to become a multistep RL algorithm suitable for more general models include averaging [10][12].

References

[1] Lin, L. J. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8:293–321, 1992.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al (2015). Human-level control through deep reinforcement learning. Nature, 518:529–533.

[3] Sutton, R. S., Szepesvari, C., Geramifard, A., and Bowling, ´M. Dyna-style planning with linear function approximation and prioritized sweeping. In International Conference on Uncertainty in Artificial Intelligence (UAI), pp. 528–536, 2008.

[4] van Seijen, H. H. and Sutton, R. S. (2015). A Deeper Look at Planning as Learning from Replay. Proceedings of the 32nd International Conference on Machine learning (ICML).

[5] van Seijen, H. H. and Sutton, R. S. (2014). True online TD(λ). In Proceedings of the 31st International Conference on Machine learning (ICML).

[6] Sutton, R. S. and Barto, A. G. (2017). Reinforcement Learning: An Introduction Complete Draft. 2nd Edition, Acessed online, MIT Press, Cambridge.

[7] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). Neuro-Dynamic Programming. Athena Scientific.

[8] Dayan, P. (1992). The convergence of TD(λ) for general λ. Machine Learning, 8(3):341–362.

[9] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1):9–4

[10] van Hasselt, H. and Sutton, R. S. (2015). Learning to predict independent of span. arXiv:1508.04582.

[11] Lin, L. J. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning, 8:293–321, 1992.

[12] Sutton, R. S. TD models: Modeling the world at a mixture of time scales. In Proceedings of the 12th International Conference on Machine Learning (ICML), pp. 531–539, 1995.