True Online TD-Replan(λ) Method Planning through Replaying

Abdulrahman Altahhan, *Member, IEEE*

***Abstract*—In this paper, we develop a new planning method that extends the capabilities of the true online TD to allow an agent to efficiently replay all or part of its past experience, online in the sequence that they appear with, either in each step or sparsely according to the usual λ parameter. In this new method that we call True Online TD-Replan(λ), the λ parameter plays a new role in specifying the density of the replay process in addition to the usual role of specifying the depth of the target's updates. We demonstrate that, for problems that benefit from experience replay, our new method outperforms true online TD(λ), albeit quadratic in complexity due to its replay capabilities. In addition, we demonstrate that our method outperforms other methods with similar quadratic complexity such as Dyna Planning and TD(0)-Replan algorithms. We test our method on two benchmarking environments, a random walk problem that uses simple binary features and on a myoelectric control domain that uses both simple sEMG features and deeply extracted features to showcase its capabilities.**

***Index Terms*—TD, TD(λ), true online TD with Replay, Replan, Replanning, experience replay.**

I. Introduction

EXPERIENCE replay plays an important role in the context of reinforcement learning algorithms. In this paper we tackle the issue of building a robust method that allows the agent to maximize its experience replay capability with relatively cheap complexity. We will tackle multi-step sequential replay algorithm where the agent replays a sequence of past experience steps in the order they appeared with. This issue have been partially attempted in [1] where the algorithm used TD(0) update rules as its basis. In this work, we will extend the ideas developed in [1] to the true online TD(λ) updates. In particular, we will build a new method based on three requirements. First, we would like to be able to utilise a multi-step targets for each replay update instead of the one step target update of TD(0), this allows the method to choose how deep its targets are going to be for each replay update. The second requirement, is that we want to allow the algorithm designer to choose how much of the past experience updates he/she needs to incorporate and how frequently this replay process will be performed. Thirdly, regardless of the replay depth and the target's depths, the method should be efficient even on the limit when the replay depth is maximal. To achieve these goals we first introduce a method, namely online λ-return TD-Replan, that takes experience replay to its extreme by allowing the agent to replay all of its past experience online in every time step. Unlike previous work, this method allows us to utilize the multi-step interim λ-return targets for each replay update instead of the one-step target of TD(0). We show how to deduce an online efficient incremental method, namely TD(λ)-Replan, that is equivalent to online λ-return TD-Replan, but has a complexity that is not related to the time step, we prove the equivalency mathematically in Theorem 1. We show then that the true online TD algorithm becomes a special case of the true online TD-Replan algorithm in Theorem 2. We then soften the replay maximal replay requirement by utilizing a parameterization that interpolates between no-replay and replay-all, effectively allowing the designer to choose the online replay depth similar to how we choose the depth of λ-return in the true online TD(λ). Finally, the target's depth and the replay process depth of the resultant TD(λ)-Replan($\acute{\lambda}$) method, can be aligned to formulate TD-Replan(λ).

Reinforcement learning deals with Markov Decision Process $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ where we have a set of discrete actions $\mathcal{A}$ that are available for the agent to pick from at each state s. We denote the feature representation of state s as $\boldsymbol{\phi}(s)$ and the number of features $|\boldsymbol{\phi}| = n$. $r(s, a, \acute{s})$ is the expected reward signal for taking action a at state s then moving to state $\acute{s}$. The feature representation can be complex. For example, the features can be obtained from a deep architecture such as from the encoder of an auto-encoder [2] or a set of stacked auto encoders [3] . We can then employ whatever algorithm we have on the resultant features [4]. In this case each learning stage is dissected from the other stages. In this paper we take this approach due to its simplicity and theoretical guarantees. Alternatively, one can take a more integrated end-to-end learning approach. An example of such integrated approach is to build a deep feature extraction model and append a fully connected layer with linear activation function, we then can backpropagate the error coming out of the last layer further into previous layers [5]. However, despite the impressive empirical achievement of such models convergence guarantees do not apply straight on a non-linear model [4]. The high performance in the second approach can mainly be attributed to the stability and agility provided by the replay process [6]. By building the targets in the method itself we conjecture that we can avoid having to use a separate network to represents the targets as in [5]. The better performance of an intermediate replay depth value can be attributed to the ability to balance the importance of sequence of events and to break the strong correlation that can lead to instability when non-linear function approximation is used. In this work we build an algorithm that is particularly suitable for deep learning whether the learning process is dissected or integrated.

A. Altahhan is with the School of Built Environment, Engineering and Computing, Leeds Beckett University, Leeds, UK, LS6 3QS. e-mail: a.altahhan@leedsbeckett.ac.uk.

Throughout our presentation we will assume that the behaviour policy is derived directly from the learning weights for action-values and we do not consider policy gradient methods. Our focus will be on multi-steps policy evaluation methods that involves looking back at multiple updates for steps that happened in the past and repeat their respective updates, i.e. replay them.

II. Bundled Experience Replay

Replay can be categorized as sequential with specific frequency (ex. replaying past sequence of 10 steps every other step), which is the topic of this paper, and non-sequential, for example the one used in [7]. When sequential replay is incorporated in the method we can systematically reduce its complexity to be related to the number of used features, whether a deep or a shallow model is used, rather than in the number of time steps of an experience. More recently, the replay process has been taken from a different perspective through the work of [8] to mean re-evaluating past target updates using what has been recently learned by the agent, they tagged it as replay. In this sense their algorithms redo the same set of past updates, with the same initial learning weights in each step, using updated targets, in order to benefit from past experience. In their setting each bundle of updates always starts from the same initial weight's values, which is a key issue. Although this simplifies finding incremental forms for the learning process, however in that sense their approach is more of reevaluating the target rather than actually replaying the experience. From our perspective, replaying the experience requires going through a bundle of past experience and redo the updates as if the agent went through them again but with its current set of weights [1]. Therefore, our approach for experience replay is more like the original approach of Lin (combined with the notion of target revaluation) but contrary to Lin's it is more sequential and more intensive to promote learning agility. From a learning perspective, each time a new online interaction takes place between the agent and the environment, the replay process should allow the model to start with a better initialization of the weights.

III. TD-Replan with Interim λ-return

Contrary to [1] we use interim λ-return as the target for each update. Interim λ-returns takes advantage of all past experience to obtain a more accurate estimate of the targets of the TD updates. In this section we show the forward view of our elaborate replay method using interim λ-returns similar to the way true online TD(λ) was constructed [9]). The forward true online TD() algorithm is largely kept as is with one important change. We will run through all past updates and redo them all as a bundle, using the latest model weights, i.e without reinitializing them back to their original values. We assume that in each time step t, the algorithm is going to go back to all past steps trajectory and replay every single update based on its latest weights. Index t will be used to represent current time step, while index k will be used to represent past steps, where $0 \leq k \leq t$. For example, the model's weights at time step t that are used to replay past step k are denoted $\boldsymbol{\theta}_k^t$, while the weights that are the results of replaying past time step k are denoted $\boldsymbol{\theta}_{k+1}^t$. $\boldsymbol{\theta}_i^i$ will be abbreviated as $\boldsymbol{\theta}_i$; i.e. $\boldsymbol{\theta}_i := \boldsymbol{\theta}_i^i$, for example when we see $\boldsymbol{\theta}_t$ it stands for $\boldsymbol{\theta}_t^t$. We will devote our attention in this section to the last layer of the model that is used to represents the value function V. Each one of the forward TD replay updates can be written as

$$\boldsymbol{\theta}_{k+1}^{t+1} = \boldsymbol{\theta}_k^{t+1} + \alpha_k \nabla_{\boldsymbol{\theta}} V \left(G_k^{\lambda|t+1} - V(s|\boldsymbol{\theta}_k^{t+1}) \right) \tag{1}$$

where $G_k^{\lambda|t+1}$ is the interim λ-return introduced in [9] and is defined as:

$$G_k^{\lambda|t} = \sum_{i=1}^{t-k-1} \lambda^{i-1} G_k^{(i)} + \lambda^{t-k-1} G_k^{(t-k)} \tag{2}$$

$$G_k^{(i)} = (1-\lambda) \sum_{j=1}^{i} \gamma^{j-1} R_{k+j} + \gamma^i V(S_{k+j}|\boldsymbol{\theta}_{k+j-1}) \tag{3}$$

Note that when $k = t-1$ then $G_k^{\lambda|t} = G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}|\theta_t)$ which is the usual one-step target of TD(0). We assume that the last layer is linear, hence a linear model is used to express the value function, $V(s|\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s)$, where $\nabla_{\boldsymbol{\theta}} V = \boldsymbol{\phi}(s)$. This assumption entails some restriction but it does not prevent us from using a non-linear and complex layers that come before this last layer in order to build a full fledged deep learning model.

In this case, the set of the replay updates (1) is written as

$$\boldsymbol{\theta}_{k+1}^{t+1} = \boldsymbol{\theta}_k^{t+1} + \alpha_k \boldsymbol{\phi}_k \left(G_k^{\lambda|t+1} - \left(\boldsymbol{\theta}_k^{t+1}\right)^\top \boldsymbol{\phi}_k \right) \tag{4}$$

$$\boldsymbol{\theta}_{k+1}^{t+1} = \boldsymbol{A}_k \boldsymbol{\theta}_k^{t+1} + \boldsymbol{b}_k^t \tag{5}$$

$$\boldsymbol{A}_k := \left[\boldsymbol{\mathcal{I}}_{n\times n} - \alpha_k \boldsymbol{\phi}_k \boldsymbol{\phi}_k^\top \right] \tag{6}$$

$$\boldsymbol{b}_k^t := \alpha_k \boldsymbol{\phi}_k G_k^{\lambda|t+1} \tag{7}$$

where $\boldsymbol{A}_k$ is a squared matrix, $\boldsymbol{b}_k^t$ is a vector and n is the number of weights used to encode the value function. The time and space complexity of the above algorithm can be made reasonable and be only related to n. Although each step is entailing t updates with complexity $O(t \times n)$, we shall use the formalism used by [10] and [9] to make the complexity $O(n^2)$ assuming that $t > n$ or $t \gg n$ in most of the cases.

In addition, it can be proven [9] that:

$$\begin{aligned} G_k^{\lambda|t+1} - G_k^{\lambda|t} &= (\lambda\gamma)^{t-k} (R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_{t+1} - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t) \\ G_k^{\lambda|t+1} &= G_k^{\lambda|t} + (\lambda\gamma)^{t-k} \delta_t' \end{aligned} \tag{8}$$

Let us see how the algorithm behaves along with the targets $G_k^{\lambda|t+1}$ in the below snippet.

$$\begin{aligned} \boldsymbol{\theta}_1^4 &= \boldsymbol{A}_0 \boldsymbol{\theta}_3^3 + \boldsymbol{b}_0^4 : \quad \boldsymbol{A}_0 = \boldsymbol{\mathcal{I}} - \alpha_0 \boldsymbol{\phi}_0 \boldsymbol{\phi}_0^\top, \quad \boldsymbol{b}_0^4 = \alpha_0 \boldsymbol{\phi}_0 G_0^{\lambda|4} \\ \boldsymbol{\theta}_2^4 &= \boldsymbol{A}_1 \boldsymbol{\theta}_1^4 + \boldsymbol{b}_1^4 : \quad \boldsymbol{A}_1 = \boldsymbol{\mathcal{I}} - \alpha_1 \boldsymbol{\phi}_1 \boldsymbol{\phi}_1^\top, \quad \boldsymbol{b}_1^4 = \alpha_1 \boldsymbol{\phi}_1 G_1^{\lambda|4} \\ \boldsymbol{\theta}_3^4 &= \boldsymbol{A}_2 \boldsymbol{\theta}_2^4 + \boldsymbol{b}_2^4 : \quad \boldsymbol{A}_2 = \boldsymbol{\mathcal{I}} - \alpha_2 \boldsymbol{\phi}_2 \boldsymbol{\phi}_2^\top, \quad \boldsymbol{b}_2^4 = \alpha_2 \boldsymbol{\phi}_2 G_2^{\lambda|4} \\ \boldsymbol{\theta}_4^4 &= \boldsymbol{A}_3 \boldsymbol{\theta}_3^4 + \boldsymbol{b}_3^4 : \quad \boldsymbol{A}_3 = \boldsymbol{\mathcal{I}} - \alpha_3 \boldsymbol{\phi}_3 \boldsymbol{\phi}_3^\top, \quad \boldsymbol{b}_3^4 = \alpha_3 \boldsymbol{\phi}_3 G_3^{\lambda|4} \end{aligned}$$

This is the same process performed in developing true online TD, except we will force each bundle of updates to initialise the first weight with the weight of the previous bundle of updates. By doing so we will be replaying all past updates in

each time step. It should be noted that if we fixed the initial weights of a bundle of update steps, then the algorithm turns into just a revaluation of all past rules instead of replaying past experience, this is what algorithms at [8], [10] and [9] are performing, there is no replay process utilized in them.

Hence, based on our algorithm, the final weights $\boldsymbol{\theta}_4^4$ for time steps 4 can be calculated in terms of the initial weights $\boldsymbol{\theta}_3^3$ by backward substitutions as

$$\boldsymbol{\theta}_4 = \boldsymbol{\theta}_4^4 = \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{A}_0\boldsymbol{\theta}_3^3 + \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{b}_0^4 + \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{b}_1^4 + \boldsymbol{A}_3\boldsymbol{b}_2^4 + \boldsymbol{b}_3^4$$

Note that we we need $G_k^{\lambda|4}$ to be able to calculate $\boldsymbol{b}_k^4$. It can be easily proven by induction that

$$\boldsymbol{\theta}_{t+1}^{t+1} = \left(\prod_{i=t}^{0} \boldsymbol{A}_i\right)\boldsymbol{\theta}_t^t + \sum_{k=0}^{t}\left(\prod_{i=t}^{k+1} \boldsymbol{A}_i\right)\boldsymbol{b}_k^{t+1} \tag{9}$$

It should be noted that this algorithm is completely different than the algorithm that we developed in [1] in two ways; first the basic updates of each replay step is based on true online TD not on TD(0), i.e. our algorithm uses the truncated λ-return targets, the second difference is that the matrices A_k are defined differently. It should be noted that every bundle of update ends at $\boldsymbol{\theta}_{t+1}^{t+1}$ which can be used to summarize all past sequence of intermediate updates as has been shown in [9]. We call the above algorithm the *true online λ-return TD-Replan* to emphasise two things; first that the algorithm is replaying all past experience using true online TD(λ) updates, second to emphasise the link between replaying and planning which has been already established in [8]. Clearly, this algorithm is expensive with a complexity of $O(n \times t)$ and in its current form it is impractical. In the next section we develop a more efficient way to achieve the same set of updates. This algorithm constitutes the non-incremental forward view of a more efficient and incremental algorithm that we call the *true online TD(λ)-Replan*.

IV. True Online TD(λ)-Replan: An Incremental Online View

To arrive to a correct efficient form for the intensive replay mechanism presented in the previous section, we need to extend the mathematical formulation developed in deriving the incremental forms of the true online TD(λ) to accommodate the replay process.

Theorem 1: Given a set of n weights $\boldsymbol{\theta}$ that are due to the forward true online TD(λ)-Replan algorithm shown earlier, we can obtain exactly $\boldsymbol{\theta}$ incrementally according to the following step updates

$$\begin{aligned}
\boldsymbol{A}_t &= \left[\boldsymbol{\mathcal{I}}_{n\times n} - \ \alpha_t\boldsymbol{\phi}_t{\boldsymbol{\phi}_t}^\top\right] \\
\boldsymbol{e}_t &= \boldsymbol{A}_t\gamma\lambda\boldsymbol{e}_{t-1} + \alpha_t\boldsymbol{\phi}_t \\
\bar{\boldsymbol{e}}_t &= \boldsymbol{A}_t\bar{\boldsymbol{e}}_{t-1} + \boldsymbol{e}_t\Big(\delta_t + \boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\Big) + \alpha_t\boldsymbol{\phi}_t\boldsymbol{\theta}_t^\top\boldsymbol{\phi}_t \\
\bar{\boldsymbol{A}}_t &= \boldsymbol{A}_t\bar{\boldsymbol{A}}_{t-1} \\
\boldsymbol{\theta}_{t+1} &= \bar{\boldsymbol{A}}_t\boldsymbol{\theta}_t + \bar{\boldsymbol{e}}_t
\end{aligned}$$

Proof: We start from (9), similar to [9] we denote

$$\boldsymbol{A}_t^{k+1} := \boldsymbol{A}_t\boldsymbol{A}_{t-1}\ldots\boldsymbol{A}_{k+1} = \prod_{i=t}^{k+1}\boldsymbol{A}_i \tag{10}$$

$$\boldsymbol{A}_t^{t+1} := \boldsymbol{\mathcal{I}}_{n\times n} \tag{11}$$

$$\boldsymbol{A}_t^{t} = \boldsymbol{A}_t \tag{12}$$

We define

$$\bar{\boldsymbol{A}}_t := \boldsymbol{A}_t^0 \tag{13}$$

$$\bar{\boldsymbol{e}}_t := \sum_{k=0}^{t}\boldsymbol{A}_t^{k+1}\boldsymbol{b}_k^{t+1} \tag{14}$$

Update (9) becomes

$$\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{A}}_t\boldsymbol{\theta}_t + \bar{\boldsymbol{e}}_t \tag{15}$$

Now we turn our attention to $\bar{\boldsymbol{e}}_t$ to come up with an incremental form to update it online. By plugging definition in (8) in the definition of $\bar{\boldsymbol{e}}_t$ (14) and utilizing (11) we obtain that

$$\begin{aligned}
\bar{\boldsymbol{e}}_t &:= \sum_{k=0}^{t}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k G_k^{\lambda|t+1} \\
&= \sum_{k=0}^{t-1}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k G_k^{\lambda|t+1} + \alpha_t\boldsymbol{\phi}_t G_t^{\lambda|t+1} \\
&= \sum_{k=0}^{t-1}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k\left(G_k^{\lambda|t} + (\gamma\lambda)^{t-k}\acute{\delta}_t\right) \\
&\qquad + \alpha_t\boldsymbol{\phi}_t\left(R_{t+1} + \gamma\boldsymbol{\theta}_t^\top\boldsymbol{\phi}_{t+1} \pm \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\right) \\
&= \boldsymbol{A}_t\sum_{k=0}^{t-1}\boldsymbol{A}_{t-1}^{k+1}\alpha_k\boldsymbol{\phi}_k G_k^{\lambda|t} + \acute{\delta}_t\sum_{k=0}^{t-1}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k(\gamma\lambda)^{t-k} \\
&\qquad + \alpha_t\boldsymbol{\phi}_t\left(\acute{\delta}_t + \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\right) \\
&= \boldsymbol{A}_t\bar{\boldsymbol{e}}_{t-1} + \acute{\delta}_t\sum_{k=0}^{t-1}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k(\gamma\lambda)^{t-k} + \alpha_t\boldsymbol{\phi}_t\left(\acute{\delta}_t + \boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\right)
\end{aligned}$$

$$\bar{\boldsymbol{e}}_t = \boldsymbol{A}_t\bar{\boldsymbol{e}}_{t-1} + \acute{\delta}_t\boldsymbol{e}_t + \alpha_t\boldsymbol{\phi}_t\left(\boldsymbol{\theta}_{t-1}^\top\boldsymbol{\phi}_t\right) \tag{16}$$

$$\boldsymbol{e}_t := \sum_{k=0}^{t}(\gamma\lambda)^{t-k}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k \tag{17}$$

Note that α_t is included in the definition of $\bar{\boldsymbol{e}}_t$. In addition, $\boldsymbol{e}_t$ is defined as in [9] and hence we can readily utilize its derived incremental form

$$\begin{aligned}
\boldsymbol{e}_t &:= \sum_{k=0}^{t}(\gamma\lambda)^{t-k}\boldsymbol{A}_t^{k+1}\alpha_k\boldsymbol{\phi}_k \\
\boldsymbol{e}_t &= \boldsymbol{A}_t\gamma\lambda\boldsymbol{e}_{t-1} + \alpha_t\boldsymbol{\phi}_t
\end{aligned} \tag{18}$$

Finally, from the definition of $\boldsymbol{A}_t^0$ in (10) we have

$$\bar{\boldsymbol{A}}_t = \boldsymbol{A}_t\boldsymbol{A}_{t-1}\ldots\boldsymbol{A}_0 = \boldsymbol{A}_t\bar{\boldsymbol{A}}_{t-1} \tag{19}$$

which constitutes the incremental form for $\bar{\boldsymbol{A}}_t$. The initial conditions as per the definitions are set to $\bar{\boldsymbol{A}}_{-1} = \boldsymbol{\mathcal{I}}_{n\times n}$, $\bar{\boldsymbol{A}}_{-1} = \boldsymbol{\mathcal{I}}_{n\times n},$, $\boldsymbol{e}_{-1} = \boldsymbol{0}_{n\times 1}$ which yields TD(0) update for $t = 0$. Hence our algorithm is defined by (7), (15), (16), (18) and (19) which conclude our proof. ■

V. Efficient TD-Replan Algorithm

By substituting A_t and reorganising the terms so that we have vector $\times$ matrix multiplication and not matrix $\times$ matrix multiplication we obtain the true online TD(λ)-Replan method shown below:

$$\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \alpha_t \boldsymbol{\phi}_t \left(1 - \boldsymbol{e}_{t-1}^\top \boldsymbol{\phi}_t\right) \tag{20}$$

$$\bar{\boldsymbol{e}}_t = \bar{\boldsymbol{e}}_{t-1} + \boldsymbol{e}_t\Big(\delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\Big) - \alpha_t \boldsymbol{\phi}_t \left(\bar{\boldsymbol{e}}_{t-1}^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\right) \tag{21}$$

$$\bar{\boldsymbol{A}}_t = \bar{\boldsymbol{A}}_{t-1} - \alpha_t \boldsymbol{\phi}_t \left(\boldsymbol{\phi}_t^\top \bar{\boldsymbol{A}}_{t-1}\right) \tag{22}$$

$$\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{A}}_t \boldsymbol{\theta}_t + \bar{\boldsymbol{e}}_t \tag{23}$$

Formulating this method as a learning episodic algorithm for prediction is given in Algorithm 1.

Algorithm 1 TD(λ)-Replan(1) Learning

Input: $\alpha, \gamma, \lambda, \boldsymbol{\theta}_{init}$
Output: $\boldsymbol{\theta}$

```
obtain initial φ
θ ← θ_init
for all episodes do
  e ← 0, ē ← 0, Ā ← I, V_old ← 0
  while S is not Terminal do
    obtain next feature vector φ' and reward R
    V ← θ^⊤ φ
    V' ← θ^⊤ φ'
    δ ← R + γV' − V
    e ← eγλ − αφ(γλe^⊤ φ − 1)
    ē ← ē − αφ(ē^⊤ φ − V_old) + e(δ + V − V_old)
    Ā ← Ā − αφ(φ^⊤ Ā)
    θ ← Āθ + ē
    V_old ← V'
    φ ← φ'
  end while
end for
```

VI. True Online TD(λ) as a Special Case of True Online TD(λ)-Replan

In this section we show that the true online TD(λ) can be viewed as a special case of the true online TD(λ)-Replan by fixing the weights used in the update rule (15).

Theorem 2: When $\theta_t = \theta_0$ the true online TD(λ)-Replan algorithm reduces to the usual linear true online TD(λ).

Proof: By defining $\bar{\boldsymbol{a}}_t := \bar{\boldsymbol{A}}_t \boldsymbol{\theta}_0$, rule (23) becomes:

$$\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{A}}_t \boldsymbol{\theta}_0 + \bar{\boldsymbol{e}}_t \tag{24}$$

$$\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{a}}_t + \bar{\boldsymbol{e}}_t \tag{25}$$

In addition, $\bar{A}_t$ can be vectorized into $\bar{a}_t$ by multiplying (22) by θ_0 and substituting by the $\bar{\boldsymbol{a}}_t$ definition:

$$\bar{\boldsymbol{a}}_t = \bar{\boldsymbol{a}}_{t-1} - \alpha_t \boldsymbol{\phi}_t \left(\bar{\boldsymbol{a}}_{t-1}^\top \boldsymbol{\phi}_t\right) \tag{26}$$

Equation (26) can be combined with (21) by simple addition of $\bar{\boldsymbol{a}}_t + \bar{\boldsymbol{e}}_t$ and substituting by $\boldsymbol{\theta}_{t+1}$ and substituting $\bar{\boldsymbol{a}}_{t-1} + \bar{\boldsymbol{e}}_{t-1}$ by $\boldsymbol{\theta}_t$ we have

$$\begin{aligned}\bar{\boldsymbol{e}}_t + \bar{\boldsymbol{a}}_t &= \bar{\boldsymbol{e}}_{t-1} + \bar{\boldsymbol{a}}_{t-1} \\ &\quad - \alpha_t \boldsymbol{\phi}_t \left(\bar{\boldsymbol{a}}_{t-1}^\top \boldsymbol{\phi}_t\right) + \boldsymbol{e}_t\Big(\delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\Big) \\ &\quad - \alpha_t \boldsymbol{\phi}_t \left(\bar{\boldsymbol{e}}_{t-1}^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\right) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{e}_t\Big(\delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\Big) \\ &\quad - \alpha_t \boldsymbol{\phi}_t \left(\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\right)\end{aligned} \tag{27}$$

Update (27) along with (20) correspond exactly to the update rules of the true online TD(λ) algorithm which conclude our proof. ■

The TD(λ)-Replan(1) replays all past experience in every step and as we said earlier it is on the far end of the spectrum of replaying. In order to reach a compromise that allows our algorithm to represents all range of replaying from non to full, we will look into reducing the complexity of our algorithm. So far, we have replayed all past experience by having current step replay rules to use past step weights as a base to update to current weights. However, our method allows for more flexibility.

VII. True Online TD-Replan(λ)

The above methods constitutes the two ends of the spectrum of experience replay. The next normal development is to explore ways of representing all the spectrum between these two ends. In other words, we would like to construct a method that allows us to specify how much of a replay, if any, the agent should experience in each online step. This is an open question that one can address in several ways. One way to perform this requirement is by constructing a linear combination of the two methods using a hyper parameter $\acute{\lambda}$. Or, we can use the same λ parametrisation in order to combine the depth of the target and the depth of the replay in one hyper parameter. To achieve this, we can alter the TD(λ)-Replan method:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \bar{\boldsymbol{A}}_t \left(\acute{\lambda}\boldsymbol{\theta}_t + \left(1 - \acute{\lambda}\right)\boldsymbol{\theta}_0\right) + \bar{\boldsymbol{e}}_t \\ &= \bar{\boldsymbol{A}}_t \left(\acute{\lambda}\boldsymbol{\theta}_t + \left(1 - \acute{\lambda}\right)\boldsymbol{\theta}_0\right) + \left(\acute{\lambda}\bar{\boldsymbol{e}}_t + \left(1 - \acute{\lambda}\right)\bar{\boldsymbol{e}}_t\right) \\ &= \acute{\lambda}(\bar{\boldsymbol{A}}_t \boldsymbol{\theta}_t + \bar{\boldsymbol{e}}_t) + (1 - \acute{\lambda})(\bar{\boldsymbol{A}}_t \boldsymbol{\theta}_0 + \bar{\boldsymbol{e}}_t)\end{aligned} \tag{28}$$

Since the last update is written as a linear combination of updates (15) and (24) the resultant method is a parametrized combination of the two methods: the true online TD(λ)-Replan and the true online TD(λ). Therefor, the final method that we call TD(λ)-Replan($\acute{\lambda}$) is defined by equations(7), (28), (16), (18) and (19). Below we show a policy evaluation algorithm that is based on true online TD(λ)-Replan($\acute{\lambda}$) method.

Algorithm 2 true online TD(λ)-Replan($\acute{\lambda}$) Learning

Input: $\alpha, \gamma, \lambda, \acute{\lambda}, \boldsymbol{\theta}_{init}$ {λ may $= \acute{\lambda}$}
Output: $\boldsymbol{\theta}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$
for all episodes **do**
  obtain initial state ϕ
  $\boldsymbol{\theta}_0 \leftarrow \boldsymbol{\theta}, \boldsymbol{e} \leftarrow \mathbf{0}, \bar{\boldsymbol{e}} \leftarrow 0, \bar{\boldsymbol{A}} \leftarrow \mathcal{I}, V_{old} \leftarrow 0$
  while S is not Terminal **do**
    obtain next feature vector $\acute{\phi}$
    $V \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\phi}$
    $\acute{V} \leftarrow \boldsymbol{\theta}^\top \acute{\boldsymbol{\phi}}$
    $\delta \leftarrow R + \gamma \acute{V} - V$
    $\boldsymbol{e} \leftarrow \boldsymbol{e}\gamma\lambda - \alpha\boldsymbol{\phi}(\gamma\lambda \boldsymbol{e}^\top \boldsymbol{\phi} - 1)$
    $\bar{\boldsymbol{e}} \leftarrow \bar{\boldsymbol{e}} - \alpha\boldsymbol{\phi}(\bar{\boldsymbol{e}}^\top \boldsymbol{\phi} - V_{old}) + \boldsymbol{e}(\delta + V - V_{old})$
    $\bar{\boldsymbol{A}} \leftarrow \bar{\boldsymbol{A}} - \alpha\boldsymbol{\phi}(\boldsymbol{\phi}^\top \bar{\boldsymbol{A}})$
    $\boldsymbol{\theta} \leftarrow \bar{\boldsymbol{A}} \left(\acute{\lambda}\boldsymbol{\theta} + (1 - \acute{\lambda})\boldsymbol{\theta}_0 \right) + \bar{\boldsymbol{e}}$
    $V_{old} \leftarrow \acute{V}$
    $\boldsymbol{\phi} \leftarrow \acute{\boldsymbol{\phi}}$
  end while
end for

It should be noted that when true online TD(λ)-Replan($\acute{\lambda} = 0$) is exactly equivalent to the true online TD(λ). It should be noted also that when we have $\acute{\lambda} = \lambda$ then the method is simply called TD-Replan(λ). Similar to true online TD(λ), TD-Replan(0) method is equivalent to TD(0).

VIII. TD-REPLAN APPLIED ON RANDOM WALK

In this section we show the prediction performance of our algorithms on a random walk task for benchmarking. Random walk isolates the effect of the dynamic of the environment since selecting the actions is randomised based on a probability distribution that represents a fixed policy. This allows us to concentrate on the prediction capability of an algorithm. Our environment consists of 17 states [4] and the process starts always form the far left hand side state and the episodes ends when the process reaches the far right state. In each step the action that moves the current state towards the right final state are given a fixed reward of $1/n$ where n is the number of non terminal states. While, the action that moves the current state away from the right final state are given a reward of $-1/n$, the action that takes the process to the right final state is given a 0 reward, staying in the far left state is also given a reward of 0. Both actions have the same probability and no discount is used, i.e. $\gamma = 1$. These setting allowed the RMSE error to be bounded to 1, and further allowed us to benchmark with other random walk problems. It can be easily proven that the sum of the rewards of each state can be analytically calculated to be $V(S_i) = (i-1)/n$.The features used are simple basis binary features that represents each state as a vector of zeros with one feature on at a time. Our experiments shows that our new methods TD-Replan have the least sensitivity to the step size and almost always guarantees convergence with maximum speed (in terms of number steps needed to converge). Fig. 2 shows that our algorithm outperforms the true online TD(λ) for all λ values in this domain. It also

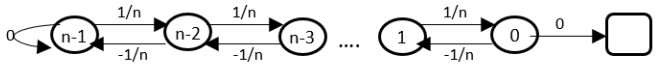

Fig. 1. Random Walk Task

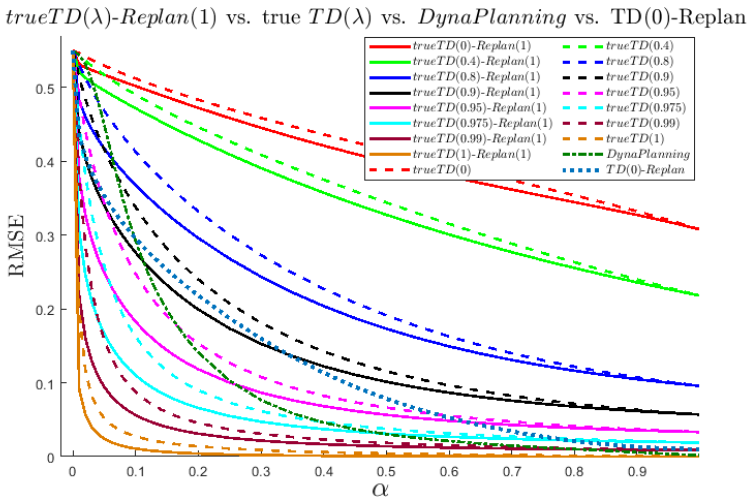

Fig. 2. Comparison of true online TD(λ)-Replan(1) with true online TD(λ), as well as TD(0)-Replan and Dyna Planning, on Random Walk on 17 of the first 10 episodes averaged over 20 trials for binary features. This shows the clear edge that our new method have over other methods despite the simplicity of the problem.

outperformed the TD(0)-Replan(1) algorithm [1] as well as the Dyna Planning algorithm [11] both of which have a similar quadratic complexity, which shows that our algorithm clearly outperform those planning algorithms as well.

IX. DEEP SPARSE VARIATIONAL AUTOENCODER AND TD-REPLAN APPLIED ON MYOELECTRIC CONTROL

In [12] authors have shown how to control a two-dimension cursor via a set of surface electromyographic (sEMG) signals obtained from forearm activities. Fourteen abled-body subjects were studied one of which has a congenital upper-limb deficiency. They have conducted a set of myoelectric control experiments; their aim was to study the effect of arm position and donning/doffing of a textile hose that they used to obtain a set of sEMG signal readings. In their experiments, each subject controlled the cursor using a set of sixteen sEMG sensory signals attached to the subject's forearm. The task is to move a cursor from a location inside a circle that is presented to the subject on a computer screen to the centre of the circle using the sEMG signals coming from their muscles activities. In each experiment a set of sixteen pre-specified locations were randomly selected one after the other and the subject has to move the cursor to the requested location. The tasks were performed in approximation and sometimes the subjects failed to reach the target position in the allocated time. The subject is considered to hit the target if he/she sustained the cursor in a radius of 0.15 for 1 second. After the subject hits or misses the target, a pause of 1 second is enforced and the

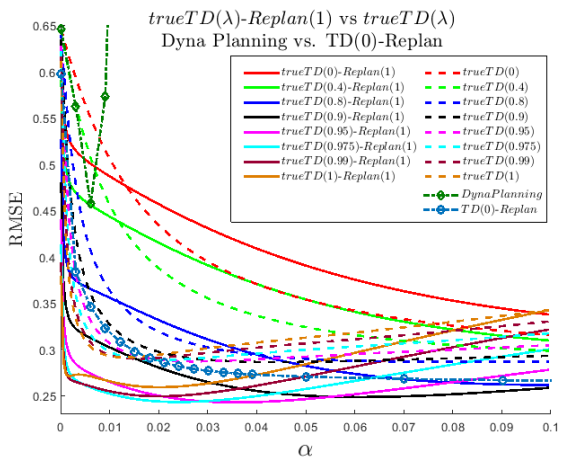

Fig. 3. Comparison of the RMSE for true online TD(λ)-Replan(1) and true online TD(λ) applied on Myoelectric control of cursor on a screen using normalised sEMG as features, all taken for the first 10 episodes and averaged over 66 trials with α values that spans 0.001 to 0.1 with 0.005 steps. the figure clearly shows that for high λ values TD(λ)-Replan(1) is more advantageous than true TD(λ). We note that our algorithm has a wider maximal area and converges quicker to an optimal performance for a wider range of learning steps α, making it more reliable and stable. Note that Dyna Planning has struggled to learn the environment's dynamics due to deep learning mapping the sEMG into a more elaborate but sparse space. On the other hand, TD(0)-Replan has performed relatively good as expected but could not outperform true TD(0.9)-Replan onward.

cursor is returned to the centre until all 16 targets have been presented (in random order). The dataset is publicly available in [13].

Our aim in this study is to show how to directly predict the future position of the cursor based on current raw sEMG arm signals, hence simplifying and transforming the ways in which such control models are constructed and trained. We will use our algorithms to predict the next position of the cursor on a screen based on the sEMG signals. The sEMG signal is packed with noise and variation that makes the task challenging. It has been attempted to be tackled using deep learning approach as in [6] but in a supervised learning settings. In addition, we conduct a comparison study to show that our algorithm prediction accuracy can considerably outperform other widely used RL planning and non-planning algorithms such as the Dyna planning and true online TD(λ) as well as TD(0)-Replan. All algorithms used the same two sets of features to compare their capacity. The first is the sEMG readings after normalisation, and the second is a set of features that are extracted from an auto encoder (AE).

In [14] authors showed that TD can predict the next sensor reading based on previous reading, they call it 'nexting'. They have shown that nexting can be performed on a large number of sensory inputs to predict their next values in parallel. Their task was a robot circling a pen continuously and their sensors (lights and ultrasonic) were predicted. In [9] authors have demonstrated how to predict two degrees of freedom task that involved the grip force and motor angle signals of a robotics hand. In this context, the sensory input plays the role of a reward. The point of view that rewards can be used to perform general prediction has been explored in several settings. For example, [15] used a variety of stimuli as a reward function to learn animal behaviour and to model conditioning.

A. Deep Auto Encoder Structure and PreTraining

The structure of the Sparse Auto Encoder is as follows. The encoder has 5 layers, the first treats the sEMG input as an image. The second, is a convolutional neural (CNN) layer that has 32 filters each of size 3x1 with a stride of 2 (yielding 8x32 features). This is followed by a relu which is then followed by another CNN layer that has a 64 filters each of size 3x1 with a stride of 2 (yielding an output of 4x64=256). This is followed by a relu layer which then is flattened into 256 neurons, which is followed by a fully connected layer to the 256 latent variables. No padding has been applied. The decoder mirrors these layers in the usual reversed manner (by using a transposed convolutional layers instead of the convolutional), both deconvolutional CNN layers have filter sizes of 2x1, no cropping has been applied. The sEMG signal has been treated as if it is a 2d gray scale image, the mission of the Sparse AE is to come up with a cleaner and a sparse decompressed representation of the sEMG signal, i.e. to map the 16 sEMG readings into $256 = 16^2$ features each specialised in a range of sEMG values. We start by training the AE in the usual unsupervised training fashion to learn the best representation for the sEMG sensor readings. We used a minibatch size of 512, the transfer function for all the encoders and decoders is the logistic sigmoid and the learning rate is set to 10^{-3}. We have trained our deep learning feature extractor using all the episodes' data regardless of the positions and trials (so all sixteen positions are considered). The input was normalised by re-scaling for each component of the 16 sEMG readings. After training the AE, we use the encoder to encode all sEMG signals in a predefined number of features that corresponds to the number of latent neurons. The number of features used (without AE) is 16 and the model used one bias, while the number of latent features considered with AE is 16^2. The number of episodes for training the AE was set to 10 where the loss was around 0.05. To show the prediction capabilities of our algorithm we use a value-function approach and we do not utilise direct parametrized policy search methods and our approach is a model-free RL. When we train our algorithm we do not try to build a model for the environment dynamics since it is not necessary to predict the value function (or to do a policy search, both of these can be done without the environment model by sampling interaction between the agent and its environment). However, we will compare the performance of our TD-Replan with Dyna Planning that create a model that predicts the next state and next reward based on current state [11].

B. True Online TD-Replan Training Results

The data set has been divided into trials each trial consists 10 random episodes that belongs to the same task (starting position for the cursor), each episodes constitutes a task of moving the cursor from a start position to the centre of a circle on the screen using the subject sEMG signal. So the number of steps of each episodes varies. In order to train a network to

perform a prediction for a task, we have bundled the episodes related to each task together. The model has been left to run to the end of each episode without a stop condition to capture the full experience. The reward signal is taken to be the normalised difference between current position and the target position for the X coordinate and the Y coordinate separately. The results are shown for predicting the X signal for brevity. All episodes of the six starting positions that have significant variation along the X axes are considered. We have not used the calibration runs of the subjects (the first 9) since during calibration one of the coordinated were artificially set to 0 and the other is set to a fixed number and the cursor was not actually moved using the sEMG [16]. Runs 22 onward were not included due to the deterioration of performance of the participants due to donning/doffing effect, therefore all runs 10-21 were utilized. All experiments were done on an online fashion and all arms positions were considered equally without distinction. The maximum number of trials is 66 (11 for each task) and all of them has been utilised to obtain the averages (in our settings a trail is a set of 10 episodes). γ was set to 0.95 and we have used the same representation across the tasks.

Fig. 3 shows the comparison for the normalised sEMG features, where no deep learning feature extractor is employed. This figure shows that our algorithm performance exceeds the performance of the true online TD for any relatively high λ values (≥ 0.8) specifically at high α values ≥ 0.05.

Fig. 4 shows clearly that our algorithm outperforms the true online TD(λ) for all λ values in this domain for the deep extracted features. The figure shows that the difference between our algorithm and the other algorithms becomes more prominent, demonstrating the suitability of our algorithm to this type of deep learning extraction. We note that true online TD(λ)-Replan converges quicker to an optimal performance making it agile. Another important property to note, is that the algorithm starts almost readily with low RMSE levels when we increase the replay depth, and quickly converges to its optimal performance for small to intermediate learning step. This demonstrate that our algorithm suitable for real time and critical applications that needs minimal training and quick response. Note that λ performed best for 0.9 as is normally expected.

Fig. 5 shows that the values that keep all methods convergent are the range shown in Fig. 4 over which TD(λ)-Replan(1) outperformed all other methods. Note that Dyna Planning is included but hardly can be seen due to its divergence for α values beyond 0.002.

Fig. 6 shows the the comparison for true online TD($\lambda = 0.9$)-Replan($\acute{\lambda}$) with different $\acute{\lambda}$, demonstrating that the depth of the replay plays an important role when the features are deep and the rewards is not delayed. We should note also that true online TD(λ)-Replan(0) = true online TD(λ) as per Theorem 2 and hence the latter is actually included in the comparison. We can see the theme of a stable optimal performance for a range of α values when we increase the replay depth (i.e. when we increase $\acute{\lambda}$) up to 0.8. For $\acute{\lambda} = 1$ the algorithm increased its performance for the smaller α values but then started to decrease for relatively higher values. This type of behaviour is expected for this hyper parameter, resembling the usual λ behaviour.

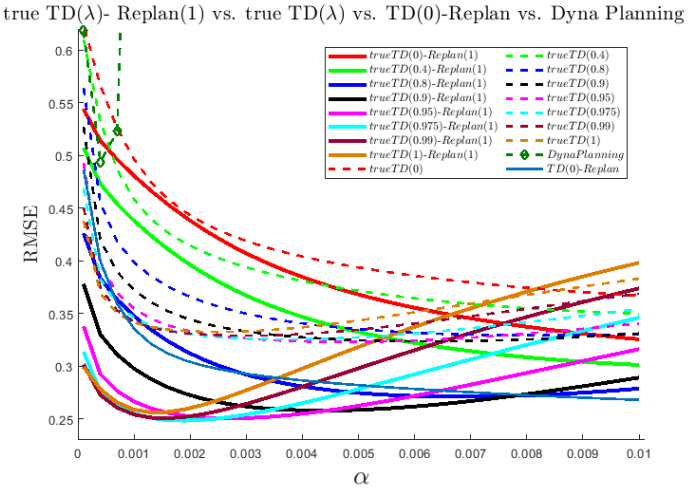

Fig. 4. RMSE comparison for true online TD(λ)-Replan(1), true online TD(λ), TD(0)-Replan and Dyna Planning. The methods are applied on myoelectric control of cursor on a screen, where 16 normalised sEMG are fed into a Sparse Auto Encoder to extract a more elaborate set of features (16^2). All results are taken for the first 10 episodes and averaged over 66 trials with α values that spans 10^{-4} to 10^{-3} with 3×10^{-4} increment. The figure clearly shows that when using deeply learned features TD(λ)-Replan(1) outperforms the true TD(λ) for all λ values with a considerable margin.

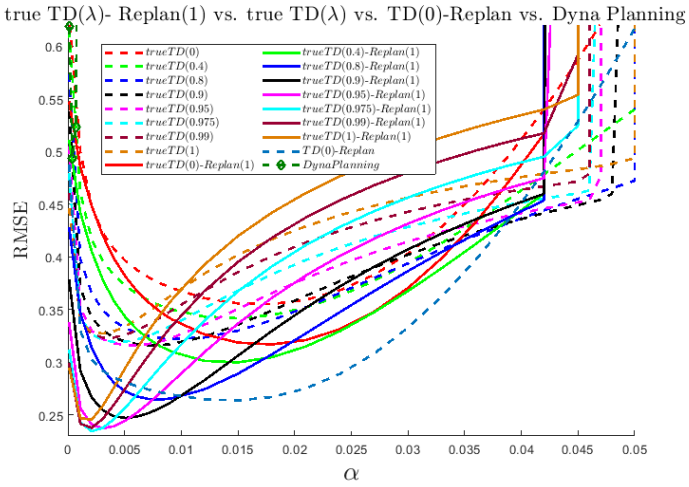

Fig. 5. Same as for Fig. 4 but over a wider range of values of α. It shows that for this type of features the smaller α values generates better results for both algorithms.

X. Conclusion

In this paper we have introduced a novel reinforcement learning method, namely the true online TD(λ)-Replan($\acute{\lambda}$) that extends the capabilities of the true online TD(λ) method to allow the method to replay all or part of its past experience according to the $\acute{\lambda}$ parameter. In addition, λ allows it to choose the depth of it targets as per norm for TD(λ) methods. The cost of the algorithm is quadratic in the number of features and the algorithm is suitable for planning. Further, we proved that the true online TD(λ) method becomes a special case of our method. This in turn allowed us to design an

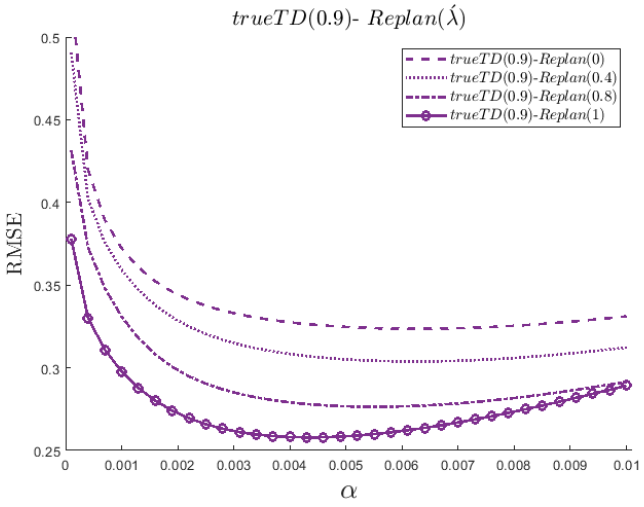

Fig. 6. RMSE Comparison for true online TD(0.9)-Replan($\acute{\lambda}$) applied on myoelectric control of cursor on a screen with a 16^2 latent features coming from an encoder, all taken for the first 10 episodes and averaged over 66 trials with α values that spans 10^{-4} to 10^{-3}. The figure clearly shows that increasing the replay depth $\acute{\lambda}$ increases the algorithm performance when using deeply learned features. Note that the same applies for other λ values but is not shown for brevity.

algorithm that can scan the full spectrum between full and partial planning based on the suitability of the application and the response and complexity requirements. Specifically, true online TD(λ) = true online TD(λ)-Replan(0). We have tested the efficacy of our algorithm on two benchmarking domains, in one of which we have combined our algorithm with a sparse autoencoder that utilises CNN layers. Both domains confirmed the utility and high performance of our algorithm in comparison to other more expensive algorithms. Further, the results shows that our algorithm constituted a good match for a deep learning extractor, paving the way for further integration in the future. Future work includes showing that our methods can be integrated with on-policy or off-policy learning update to produce new control methods, in addition to tackling an end-to-end training of a deep reinforcement learning model that is based on our method.

REFERENCES

[1] A. Altahhan, "TD(0)-replay: An efficient model-free planning with full replay," in *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, Conference Proceedings, pp. 1–7.

[2] B. Yoshua, *Learning Deep Architectures for AI*, ser. Learning Deep Architectures for AI. now, 2009. [Online]. Available: http://ieeexplore.ieee.org/document/8187120

[3] A. Ruiz-Garcia, M. Elshaw, A. Altahhan, and V. Palade, "Stacked deep convolutional auto-encoders for emotion recognition from facial expressions," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, Conference Proceedings, pp. 1586–1593.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2017.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[6] Y. Li, "Deep reinforcement learning," *ArXiv*, vol. abs/1810.06339, 2018.

[7] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992. [Online]. Available: https://doi.org/10.1007/BF00992699

[8] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *eprint arXiv:1712.01275*, p. arXiv:1712.01275, 2017. [Online]. Available: https://ui.adsabs.harvard.edu/#abs/2017arXiv171201275Z

[9] H. van Seijen, A. Rupam Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True online temporal-difference learning," *Journal of Machine Learning Research*, vol. 17, no. 145, pp. 1–40, 2016. [Online]. Available: https://ui.adsabs.harvard.edu/#abs/2015arXiv151204087V

[10] H. van Hasselt and R. S. Sutton, "Learning to predict independent of span," *CoRR*, vol. abs/1508.04582, 2015. [Online]. Available: http://arxiv.org/abs/1508.04582

[11] R. S. Sutton, C. Szepesvari, A. Geramifard, and M. P. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," *eprint arXiv:1206.3285*, p. arXiv:1206.3285, 2012. [Online]. Available: https://ui.adsabs.harvard.edu/#abs/2012arXiv1206.3285S

[12] H.-J. Hwang, J. M. Hahne, and K.-R. Müller, "Real-time robustness evaluation of regression based myoelectric control against arm position change and donning/doffing," *PloS one*, vol. 12, no. 11, pp. e0 186 318–e0 186 318, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/29095846 https://www.ncbi.nlm.nih.gov/pmc/PMC5667774/

[13] M. Atzori, A. Gijsberts, I. Kuzborskij, S. Elsig, A. M. Hager, O. Deriaz, C. Castellini, H. Müller, and B. Caputo, "Characterization of a benchmark database for myoelectric movement classification," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 1, pp. 73–83, 2015.

[14] J. Modayil, A. White, and R. S. Sutton, "Multi-timescale nexting in a reinforcement learning robot," *Adaptive Behavior*, vol. 22, no. 2, pp. 146–160, 2014. [Online]. Available: https://doi.org/10.1177/1059712313511648

[15] ——, "Multi-timescale nexting in a reinforcement learning robot," in *From Animals to Animats 12*, T. Ziemke, C. Balkenius, and J. Hallam, Eds. Springer Berlin Heidelberg, 2012, Conference Proceedings, pp. 299–309.

[16] S. Muceli, N. Jiang, and D. Farina, "Extracting signals robust to electrode number and shift for online simultaneous and proportional myoelectric control by factorization algorithms," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 22, no. 3, pp. 623–633, 2014.