White Rose Research Online

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk
Universities of Leeds, Sheffield and York

Deposited via The University of Leeds.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/id/eprint/168203/

Version: Accepted Version

Proceedings Paper:
Altahhan, A (2020) True Online TD(λ)-Replay An Efficient Model-free Planning with Full Replay. In: 2020 International Joint Conference on Neural Networks (IJCNN). 2020 International Joint Conference on Neural Networks (IJCNN), 19-24 Jul 2020, Glasgow, Scotland, United Kingdom. IEEE. ISBN: 978-1-7281-6927-9. ISSN: 2161-4393. EISSN: 2161-4407.

https://doi.org/10.1109/ijcnn48605.2020.9206608

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

UNIVERSITY OF LEEDS
University of Sheffield
UNIVERSITY of York

True Online TD(λ)-Replay An Efficient Model-free Planning with Full Replay

Abdulrahman Altahhan, *Member, IEEE*

***Abstract*—In this paper, we present a new reinforcement learning prediction method that extends the capabilities of the true online TD(λ) to allow an agent to efficiently replay all of its past experience, online in the sequence that they appear with. 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)-Replay algorithms. We showcase the capabilities of our method on two benchmarking domains, a random walk problem tested with simple binary features and on a myoelectric domain that is tested with features that are deeply extracted from sEMG signals. Experimental results confirm the particular suitability of this method for a deep architecture over other methods.**

***Index Terms*—TD, TD(λ), true online TD with Replay, full replay, 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 build a new method based on two 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 for each replay update. The second requirement is that the method should be efficiently incremental to allow a vectorised implementation without being bound to the number of past steps that will be replayed.

To achieve these goals we first introduce a method, namely the interim λ-return TD-Replay, 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 introduce an online efficient incremental method, namely the true online TD(λ)-Replay, that is equivalent to online λ-return TD-Replay, but has a complexity that is not related to the time step. Furthermore, we show that the true online TD method is a special case of the true online TD-Replay method.

We will deal with Markov Decision Process $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ to learn a value function $v_\pi(s)$ for a policy π and state s; where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space available for the agent, and $r(s, a, \acute{s})$ is the expected reward signal for executing action a at state s then transitioning to state $\acute{s}$. We denote the feature representation of state s as $\boldsymbol{\phi}(s)$ and n as the feature space dimension, i.e. $|\boldsymbol{\phi}| = n$. The feature representation can be complex. For example, the features can be obtained from an unsupervised deep architecture such as an auto-encoder [2] or a set of stacked auto encoders [3] or other similar architectures [4]. We can then employ whatever algorithm we have to train on the resultant features [5] [6]. The two learning modules; the feature extractor learning module and value-function learning module, are dissected from each other. The training can be either performed simultaneously or sequentially. For many problems, this approach is simple, helps to isolate the intrinsic RL prediction method capabilities form those of the feature extractor and has the usual theoretical convergence guarantees (with some technical conditions such as the independence of the feature's components and the learning rate diminishing property [7] [8]). In this paper we take this approach, and we train the two modules sequentially and independently. Alternatively, one can take a more end-to-end learning approach such as in [9] when the domain is more complex. However, despite the impressive empirical achievement of such models convergence guarantees do not apply straight on a non-linear model [7]. The high performance in the second approach can mainly be attributed to the stability and agility provided by the replay memory that allowed for batch updates to be used. The better performance of a random replay depth can be attributed to the ability to choose a set of trajectories and to break the strong correlation that can lead to instability when non-linear function approximation is used. However, we believe that replaying all the past experience as a block has its own advantages. In particular, it allows the agent to integrate and summarise all of its learning in each time step so that it reduces variability and dependency on the learning rate to balance out the experience. In effect, this allows the agent to be robust and to act reliably at different states. What is more, from a model perspective replaying seems particularly useful for a deep learning architecture. Our investigation confirms that an algorithm that employs replay extensively such as the true TD-Replay is empirically more suitable than other algorithms for a deep feature extractor process.

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.

II. Bundled Experience Replay

Replay can be categorized as sequential with specific frequency (ex. replaying past sequence of 10 steps every other step), which has been used in [1] and is the topic of this paper, and non-sequential, for example the one used in [10] [11] [12] [13] to mean re-evaluating past target updates using what has been recently learned by the agent, they tagged it as replay and is done in an on-policy fashion. 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 past experience updates. From our perspective, replaying the experience requires going through a bundle of past experience updates and redo them as if the agent went through them again but with its current set of weights [1]. Our intensive replaying approach can be looked at as a special case of Combined Experience Replay (CER) discussed in [14], since we include all past steps including the latest current step in each bundle of updates. The difference is that we replay all past steps and we do not sample. Albeit a special case of replay (because it is intensive and sequential), our approach for experience replay resembles the Lin's approach [15] from the sense of repeating past updates but contrary to Lin's approach it is sequential and 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-Replay 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 use the forward view of our elaborate replay method using interim λ-returns similar to the way true online TD(λ) was constructed [16]). The forward true online TD(λ) 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 will go back to all past step trajectories 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 represent the value function V. Each one of the forward TD replay updates can be written as

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

where $G_k^{\lambda|t+1}$ is the interim λ-return introduced in [16] and is defined as:

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

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

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 deep learning model. Learning can take place in two ways. The first, is performed by obtaining a good feature representation through a separate stage (by utilizing an auto-encoder for example) and then we feed these features into the last layer to obtain a policy evaluation model. We will adopt the first approach. Alternatively, model learning can be performed end-to-end in the model where the error coming from the last layer is backpropagated to previous layers.

The set of the replay updates (1) can be written as

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

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

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

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

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 [17] and [16] to make the complexity $O(n^2)$. This is useful for two reasons. First, it allows us to take advantage of the efficiency of a vectorized incremental implementation of the updates. Second, it is useful for early episodes where normally $t > n$. We will use 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 [14], [17] and [16] 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 need $G_k^{\lambda|4}$ to be able to calculate $\boldsymbol{b}_k^4$. It can be easily proven by induction that

$$\boldsymbol{\theta}_{t+1}^{t+1} = \left(\prod_{i=t}^{0} \boldsymbol{A}_i\right)\boldsymbol{\theta}_t^t + \sum_{k=0}^{t}\left(\prod_{i=t}^{k+1}\boldsymbol{A}_i\right)\boldsymbol{b}_k^{t+1} \tag{8}$$

It should be noted that this algorithm is different than the algorithm that we developed in [1] in two main aspects; first the basic updates of each replay step is based on true online TD not on TD(0), i.e. our algorithm uses the interim λ-return targets, the second difference is that the matrices A_k are defined differently.

We call the above algorithm the *interim λ-return TD-Replay* to emphasise that the algorithm is replaying all past experience using interim λ-returns. This algorithm has a built in planning capability due to the link between replaying and planning which has been already established in [13] and further confirmed in other work with model-free structures. Authors in [18], for example, confirmed that a deep neural network combined with LSTM that uses Q-learning has characteristics that are normally associated with a model-based RL planner. Our approach goes even further by making planning built in the RL method itself and decoupled form the model architecture (since we use a linear model without an LSTM or a deep neural network, although we used an autoencoder to study the suitability of our method for deeply extracted features). Other researchers have investigated the issue of planning by using a specific model architecture that enabled planning. [19] for example created a tree search architecture. Interim λ-return TD-replay algorithm constitutes the non-incremental. 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 and incremental algorithm that we call the *true online TD(λ)-replay* to perform the same set of updates.

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

To arrive at 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. This involves some considerable mathematical derivations that we omit here for simplicity and brevity. The basic ideas are to obtain a new matrix term $\bar{\boldsymbol{A}}_t$ that incorporates the replay process, and to convert the weights updates into a new eligibility trace $\bar{\boldsymbol{e}}_t$ update, both of which is initialised in each episode. Given a set of n weights $\boldsymbol{\theta}$ that are due to the forward interim λ-return TD-replay method shown earlier, we can obtain exactly $\boldsymbol{\theta}$ incrementally according to the following step updates

$$\boldsymbol{A}_t = \left[\boldsymbol{\mathcal{I}}_{n\times n} - \ \alpha_t \boldsymbol{\phi}_t \boldsymbol{\phi}_t{}^\top\right] \tag{9}$$

$$\boldsymbol{e}_t = \boldsymbol{A}_t \gamma\lambda \boldsymbol{e}_{t-1} + \alpha_t \boldsymbol{\phi}_t \tag{10}$$

$$\bar{\boldsymbol{e}}_t = \boldsymbol{A}_t \bar{\boldsymbol{e}}_{t-1} + \boldsymbol{e}_t\Big(\delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t\Big) + \alpha_t \boldsymbol{\phi}_t \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t \tag{11}$$

$$\bar{\boldsymbol{A}}_t = \boldsymbol{A}_t \bar{\boldsymbol{A}}_{t-1} \tag{12}$$

$$\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{A}}_t \boldsymbol{\theta}_t + \bar{\boldsymbol{e}}_t \tag{13}$$

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 the TD(0) update for $t = 0$. Our true online TD(λ)-Replay method is defined using the above updates.

V. Efficient True Online TD-Replay(λ) 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 a more efficient form of the true online TD(λ)-Replay method. Formulating this method as a learning episodic algorithm for prediction is given in Algorithm 1.

Algorithm 1 true Online TD(λ)-Replay 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
```

It should be noted that the true online TD(λ) can be viewed as a special case of the true online TD(λ)-Replay by fixing the weights used in the update rule (13). This is because when $\theta_t = \theta_0$ the true online TD(λ)-Replay algorithm reduces to the usual linear true online TD(λ). To see how, we define $\bar{\boldsymbol{a}}_t := \bar{\boldsymbol{A}}_t \boldsymbol{\theta}_0$, rule (13) becomes: $\boldsymbol{\theta}_{t+1} = \bar{\boldsymbol{a}}_t + \bar{\boldsymbol{e}}_t$. In addition, $\bar{A}_t$ can be vectorized into $\bar{a}_t$ by multiplying (12) by θ_0 and substituting by the $\bar{\boldsymbol{a}}_t$ definition: $\bar{\boldsymbol{a}}_t = \bar{\boldsymbol{a}}_{t-1} - \alpha_t \boldsymbol{\phi}_t \left(\bar{\boldsymbol{a}}_{t-1}^\top \boldsymbol{\phi}_t\right)$. This equation can be combined with (11) by simple addition of $\bar{\boldsymbol{a}}_t + \bar{\boldsymbol{e}}_t$ and substituting them by $\boldsymbol{\theta}_{t+1}$, and substituting $\boldsymbol{A}_t$ by its value in (7) we obtain the update rules of the true online TD(λ) algorithm: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{e}_t(\delta_t + \boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t) - \alpha_t \boldsymbol{\phi}_t(\boldsymbol{\theta}_t^\top \boldsymbol{\phi}_t - \boldsymbol{\theta}_{t-1}^\top \boldsymbol{\phi}_t)$.

VI. True Online TD(λ)-Replay Applied on Random Walk

In this section we show the prediction performance of our algorithm on a random walk task for benchmarking. Random walk is an Markov reward process (MRP) that isolates the effect of the dynamics of the environment since selecting an action is randomised based on a fixed probability distribution.

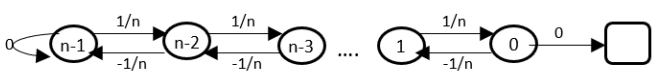

Fig. 1. The Random walk task used to benchmark our algorithm.

This allows us to examine the prediction capability of an algorithm. Fig. 1 shows our random walk task. It consists of 17 states, the process starts always form the far left hand side state and the episodes ends when the process reaches the far right state. However, we amended this MRP's rewards scheme, that is typically used, to activate the planning capabilities of our algorithm. Unlike the typical MRPs used in the literature [6] that is rewarded by 1 at the terminal state only, our process is rewarded by $1/n$ for moving the current state towards the right terminal state (but not reaching it), where n is the number of non terminal states, and it gets a reward of 0 when it moves to the terminal state. While, transitioning to the left is given a reward of $-1/n$ and staying in the far left state is given a reward of 0. Both transitions, left and right have the same probability $= 0.5$ and no discount is used, i.e. $\gamma = 1$. These settings 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 value of each state can be analytically calculated to be $V(S_i) = i/n; \quad i = 0, ..., n-1$. The features used are simple basis binary features that represents each state as a vector of zeros with one active feature at a time.

Our experiments shows that the true online TD(λ)-Replay method has the least sensitivity to the step size and almost always guarantees convergence with maximum speed (in terms of the number steps needed to converge). Fig. 2 shows that the true online TD(λ)-Replay method outperformed the true online TD(λ) method for all λ values in this domain. It also outperformed the TD(0)-Replay algorithm [1] as well as the linear Dyna Planning algorithm [20] both of which have a similar quadratic complexity. This shows that our algorithm clearly outperforms those planning algorithms as well.

VII. True Online TD(λ)-Replay Applied on Myoelectric Domain

In [21] authors have shown how to allow a subject to control a two-dimension cursor via a set of surface electromyographic (sEMG) signals obtained from the subject's forearm activities. They have used a supervised learning approach. Fourteen abled-body subjects were studied one of which has a congenital upper-limb deficiency. 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. In each experiment, a set of sixteen pre-specified cursor locations were randomly selected to the subject one after the other and the subject has to move the cursor to the center of the screen. These tasks were performed in approximation and sometimes a subject failed to do the task

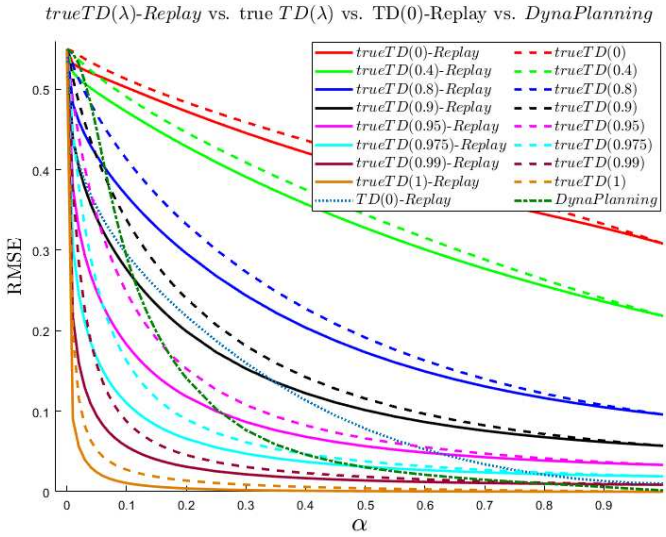

Fig. 2. Comparison of true online TD(λ)-Replay with true online TD(λ), as well as TD(0)-Replay and Dyna Planning, on 17-state random walk process, the RMSE results are averaged over 20 trials for the first 10 episodes, where binary features are used. This shows the clear edge that our new method have over other methods despite the simplicity of the problem.

in the allocated time. Each task is repeated for 33 times called runs. The data set is publicly available [22].

Our aim in this study is to show how to *predict* the future position of the cursor based on current raw sEMG arm signals, hence simplifying and transforming the ways in which such models are constructed and trained. We will use our algorithm to predict the next position of the cursor on a screen based on the sEMG signals. The sEMG signal is packed with noise with large variations in its shape and intensity between the subjects which makes the task challenging. This task has been attempted using deep learning approach as in [23] under a supervised learning settings. We conduct a comparison study to show that our algorithm's prediction accuracy can considerably outperform other widely used reinforcement learning (RL) prediction and planning algorithms such as the true online TD(λ) and Linear Dyna Planning and as well as similar replaying algorithm such as TD(0)-Replay. We conducted two sets of experiments to compare these algorithms' capabilities. The first set uses the sEMG readings directly after normalisation, and the second set uses features that are extracted from an auto encoder (AE).

In [24] 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 [16] authors have demonstrated how to predict two degrees of freedom task that involved the grip force and motor angle signals of a robotics hand but their data set is not publicly available. 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, [25] used a variety of stimuli as a reward function to learn

animal behaviour and to model conditioning.

A. Online Simulation and the Data Set

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. All experiments were performed online (so the states features and the rewards were fed in a step by step manner without a priory knowledge of the trajectories) and all arms positions were considered equally without distinction. Runs 10-21 of each subject's task are deemed useful and were utilized in training the algorithm (calibration runs and donning/doffing runs were corrupted and were excluded). The maximum number of trials is 66 (11 for each task for six tasks that vary X considerably) all of which have been utilised to obtain the RMSE 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.

B. True Online TD(λ)-Replay Training with Normalised sEMG Features

We have conducted a comparison study of true online TD(λ)-Replay with true online TD(λ) for different λ values 0, 0.4, 0.8, 0.9, 0.95, 0.97, 0.99 and 1 as well as TD(0)-Replay and Dyna Planning all of which are fed the same normalised sEMG features that are mentioned in the previous section. No deep learning feature extractor is employed. Fig. 3 shows that our algorithm performance exceeds the performance of the true online TD for any relatively high λ values (≥ 0.8) specifically at high αvalues ≥ 0.05. The results are averaged for the first 10 episodes over 66 trials with α values that spans 0.0001 to 0.001 with 0.0003 steps. The figure clearly shows that for high λ values TD(λ)-Replay 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 predict the next cursor position. On the other hand, TD(0)-Replay has performed relatively good as expected but could not outperform true TD(0.9)-Replay onward.

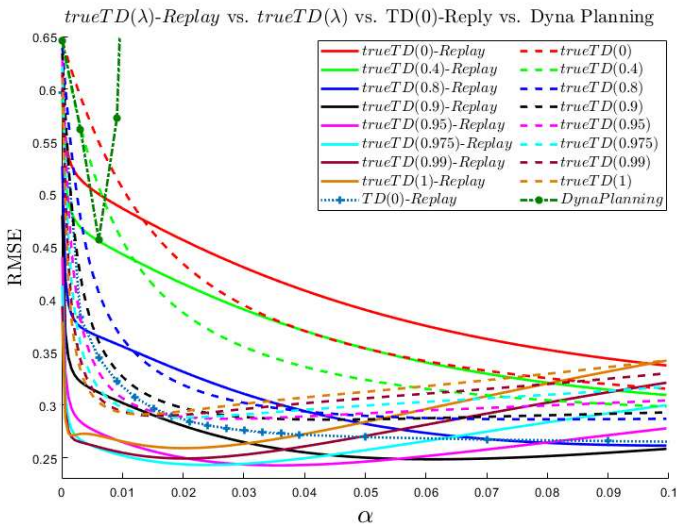

Fig. 3. RMSE comparison of true online TD(λ)-Replay, true online TD(λ), TD(0)-Replay and Dyna Planning. The methods are trained to predict the next position of a screen cursor using the normalised sEMG features. All results are averaged for the first 10 episodes over 66 trials with α values that spans 0.001 to 0.1 with 0.005 steps. The figure clearly shows that for high λ values the true online TD(λ)-Replay is more advantageous than the true online 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)-Replay has performed relatively good as expected but could not outperform the true online TD(0.9)-Replay onward.

C. Deep Auto Encoder Structure and Pre-Training

To test the suitability of our algorithms with deeply extracted features we train a deep autoencoder to extract useful features for the different RL algorithms. The training of the autoencoder is separate from the training of the RL algorithms. The structure of the Sparse Auto Encoder is as follows. The encoder has five 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 3×1 with a stride of 2 (yielding 8×32 features). This is fed into a rectified linear unit (ReLU) activation function which is then followed by another CNN layer that has a 64 filters each of size 3×1 with a stride of 2 (yielding an output of $4 \times 64 = 256$). The output of this layer is fed into a ReLU activation function then flattened into 256 neurons, which is followed by a fully connected layer to a 256 latent variables with sigmoid activation function. 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 2×1, no cropping has been applied. 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 and the learning rate is set to 10^{-3}. We have trained our deep learning feature extractor using all the available data regardless of the positions and the runs (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 256 features that corresponds to the number of latent variables. The number of epochs for training the AE was set to 10 where the loss was reduced to reach around 0.05.

D. True Online TD(λ)-Replay Training with Deeply Extracted Features

We have conducted a comparison study of true online TD(λ)-Replay with true online TD(λ) for different λ values 0, 0.4, 0.8, 0.9, 0.95, 0.97, 0.99 and 1 as well as TD(0)-Replay and Dyna Planning all of which are fed the same deeply extracted features. Fig. 4 shows this comparison. Clearly, the true online TD(λ)-Replay outperforms the true online TD(λ) for all λ values in this domain. The figure shows that the differences between our algorithm and other algorithms is more prominent than in our previous experiments with normalised sEMG features. There is a clear jump in the differences of performances between this figure the Fig. 3 demonstrating the suitability of our algorithm to this type of deep learning extraction. We note that the true online TD(λ)-Replay 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 and quickly converges to its optimal performance for a small to intermediate α (learning rate) values . This demonstrates that our algorithm is suitable for real time and critical applications that require minimal training and quick response. Note that λ performed best for 0.9 as is normally expected. Fig. 5 shows that α values that keep all methods convergent are the range shown in Fig. 4 over which TD(λ)-Replay 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.

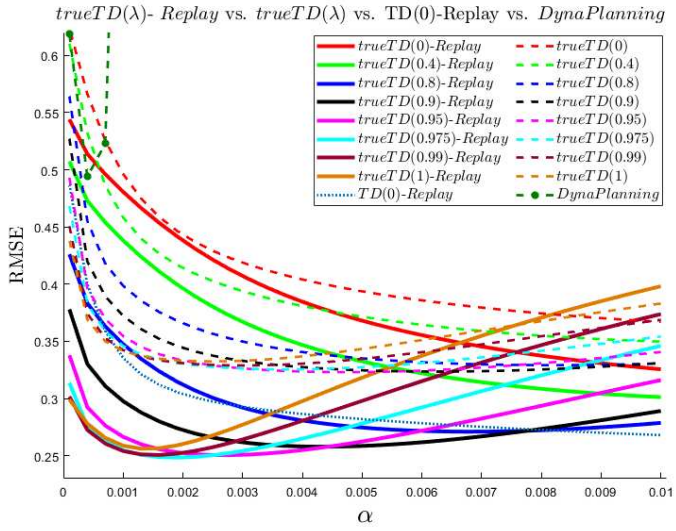

Fig. 4. RMSE comparison of true online TD(λ)-Replay, true online TD(λ), TD(0)-Replay and Dyna Planning. The methods are trained to predict the next position of a screen cursor, where 16 normalised sEMG signals are fed into a sparse auto encoder to extract a more elaborate set of features (16^2). All results are averaged for the first 10 episodes and over 66 trials with α values that spans 10^{-4} to 10^{-2} with 3×10^{-4} increment. The figure clearly shows that when using deeply learned features true online TD(λ)-Replay outperforms the true online TD(λ) for all λ values with a considerable margin.

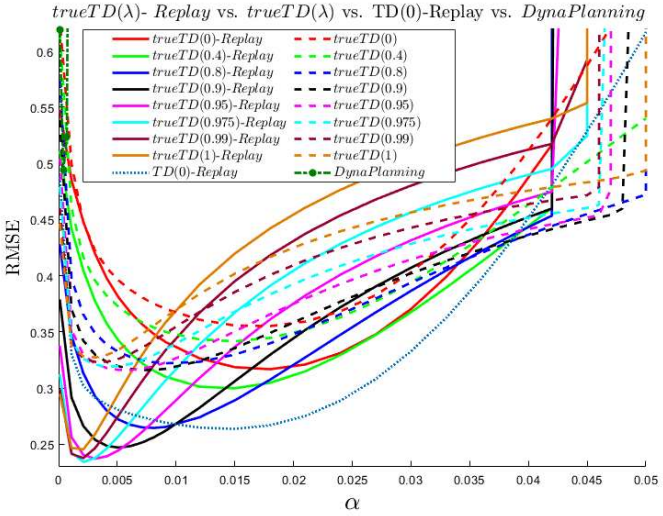

Fig. 5. Same as for Fig. 4 but over a wider range of values of α. It shows that for this type of features the smaller learning rate α values generates better results for the different algorithms.

VIII. Conclusion

In this paper we have introduced a novel reinforcement learning method, namely the true online TD(λ)-Replay that extends the capabilities of the true online TD(λ) method to allow an agent to replay all of its past experience efficiently. The parameter λ allows the agent to choose the depth of its targets as per norm for TD(λ) methods. The cost of the algorithm is quadratic in the number of weights and the algorithm is suitable for a built-in planning that is model-free. This work paves the way to design an algorithm that can scan the full spectrum between full and partial replaying ability. 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 multiple CNN layers. Both domains confirmed the utility and high performance of our algorithm in comparison to other algorithms. Further, the results shows that our algorithm constituted a good match for a deep learning extractor, paving the way for further integration and investigation in the future. Future work includes showing that our methods can be used 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 as well as parametrising the depth of the replay process.

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] A. Altahhan, "Deep feature action processing with mixture of updates," in *2015 International Conference on Neural Information Processing*,

ser. Neural Information Processing. Springer International Publishing, 2015, Conference Proceedings, pp. 1–10.

[5] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

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

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

[8] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[9] 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

[10] J. F. Hernandez-Garcia and R. S. Sutton, "Understanding multi-step deep reinforcement learning: A systematic study of the DQN target," *CoRR*, vol. abs/1901.07510, 2019.

[11] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016. [Online]. Available: http://arxiv.org/abs/1611.01224

[12] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, B. Schölkopf, and S. Levine, "Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning," *CoRR*, vol. abs/1706.00387, 2017. [Online]. Available: http://arxiv.org/abs/1706.00387

[13] H. Vanseijen and R. Sutton, "A deeper look at planning as learning from replay," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2314–2322. [Online]. Available: http://proceedings.mlr.press/v37/vanseijen15.html

[14] 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

[15] 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

[16] 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

[17] 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

[18] A. Guez, M. Mirza, K. Gregor, R. Kabra, S. Racanière, T. Weber, D. Raposo, A. Santoro, L. Orseau, T. Eccles, G. Wayne, D. Silver, and T. P. Lillicrap, "An investigation of model-free planning," *CoRR*, vol. abs/1901.03559, 2019. [Online]. Available: http://arxiv.org/abs/1901.03559

[19] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "Treeqn and atreec: Differentiable tree planning for deep reinforcement learning," *CoRR*, vol. abs/1710.11417, 2017. [Online]. Available: http://arxiv.org/abs/1710.11417

[20] 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

[21] 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/

[22] 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.

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

[24] J. Modayil, A. White, and R. 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

[25] E. A. Ludvig, R. S. Sutton, and E. J. Kehoe, "Evaluating the td model of classical conditioning," *Learning & Behavior*, vol. 40, no. 3, pp. 305–319, Sep 2012. [Online]. Available: https://doi.org/10.3758/s13420-012-0082-6