A Novel Update Mechanism for Q-Networks Based On Extreme Learning Machines

Callum Wilson
Department of Mechanical and Aerospace Engineering
University of Strathclyde
Glasgow, United Kingdom
callum.j.wilson@strath.ac.uk

Annalisa Riccardi
Department of Mechanical and Aerospace Engineering
University of Strathclyde
Glasgow, United Kingdom
annalisa.riccardi@strath.ac.uk

Edmondo Minisci
Department of Mechanical and Aerospace Engineering
University of Strathclyde
Glasgow, United Kingdom
edmondo.minisci@strath.ac.uk

***Abstract*—Reinforcement learning is a popular machine learning paradigm which can find near optimal solutions to complex problems. Most often, these procedures involve function approximation using neural networks with gradient based updates to optimise weights for the problem being considered. While this common approach generally works well, there are other update mechanisms which are largely unexplored in reinforcement learning. One such mechanism is Extreme Learning Machines. These were initially proposed to drastically improve the training speed of neural networks and have since seen many applications. Here we attempt to apply extreme learning machines to a reinforcement learning problem in the same manner as gradient based updates. This new algorithm is called Extreme Q-Learning Machine (EQLM). We compare its performance to a typical Q-Network on the cart-pole task - a benchmark reinforcement learning problem - and show EQLM has similar long-term learning performance to a Q-Network.**

I. Introduction

Machine learning methods have developed significantly over many years and are now applied to increasingly practical and real world problems. For example, these techniques can optimise control tasks which are often carried out inefficiently by basic controllers. The field of Reinforcement Learning (RL) originates in part from the study of optimal control [1], where a controller is designed to maximise, or minimise, a characteristic of a dynamical system over time. It is often impossible or impractical to derive an analytical optimal control solution for environments with complex or unknown dynamics, which motivates the use of more intelligent methods such as RL. In particular, intelligent controllers must be capable of learning quickly online to adapt to changes. The study of optimal control and RL brought machine learning into the broader field of engineering with applications to a wide variety of problems [2].

The generalisation performance of RL-derived controllers significantly improved with the incorporation of function approximators [3]. Unlike the earlier tabular methods, architectures such as fuzzy logic controllers [4] or more commonly Neural Networks (NNs) can exploit similarities in areas of the state space to learn better policies. This comes at a cost: NNs usually take a long time to train and in general they do not guarantee convergence. Furthermore, nonlinear function approximators can be unstable and cause the learning algorithm to diverge [5]. Despite this, through careful selection of hyperparameters and use of additional stability improvement measures, as will be discussed later, such function approximators can still obtain useful solutions to control problems. Of all the algorithms available for tuning network weights, backpropagation is the most widely used in state-of-the-art systems [6], [7], [8], [9]. The most common alternatives to this approach involve evolutionary algorithms, which can be used to evolve network weights or replace the function approximator entirely [10]. Such algorithms tend to show better performance but have a much higher computational cost which can make them infeasible for certain learning problems.

Extreme Learning Machines (ELMs) are a class of neural networks which avoid using gradient based updates [11]. For certain machine learning problems, ELM has several advantages over other update rules - mainly that it can be considerably faster than iterative methods for optimising network weights since they are instead calculated analytically. ELM has seen many improvements and adaptations allowing it to be applied to a wide variety of problems involving function approximation [12]. These include applications within the realm of RL, such as using a table to provide training data for an ELM network [13], approximating system dynamics using ELM to later apply RL methods [14], or using ELM theory to derive an analytical solution for weight updates based on the loss function of gradient-based updates [15]. Here we aim to use ELM in a conventional RL algorithm by only altering the neural network update rule. The algorithm which uses ELM in this manner is referred to here as the "Extreme Q-Learning Machine" (EQLM).

In this paper we develop the EQLM algorithm and compare its performance to a standard Q-network of the same complexity. The type of Q-network used here is relatively primitive but

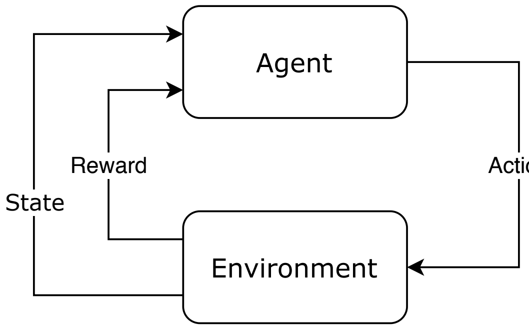

Fig. 1. Agent-environment interaction in reinforcement learning, where the agent observes a state and reward signal from the environment and uses this information to select an action to take

incorporates some features to improve stability and general learning performance to provide a reasonable comparison. A full stability analysis of each algorithm is outwith the scope of this paper, however we compare their performance using standard measures of learning. EQLM uses an incremental form of ELM which allows updates to be made online while the RL agent is interacting with the environment. Tests are carried out on a classical RL benchmark known as the cart-pole problem.

II. Background

A RL process consists of an agent which senses an environment in a certain state and carries out actions to maximise future reward [2]. The only feedback the agent receives from the environment is a state signal and reward signal and it can only affect the environment by its actions as shown schematically in Figure 1. The objective is then to maximise the total discounted reward it receives. One method for optimising the reward is Q-Learning, where the agent learns the action-value function, Q of its policy and uses this to improve the policy in an iterative process [16]. The temporal-difference (TD) error of Q is defined as shown:

$$e = Q(s_t, a_t) - \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a)\right) \tag{1}$$

where s_t, a_t, and r_t denote state, action, and reward respectively at time-step t, γ is the discount factor which determines the affect of long term rewards on the TD error, and $Q(s,a)$ is the estimated action-value function. We approximate $Q(s,a)$ using a feed-forward NN with parameters θ and, for the standard Q-network, perform updates using the mean-squared TD error. Approximating the value function and updating using TD-error forms the basis for the most rudimentary Q-Learning algorithms. This section details the additional features of the Q-Learning algorithm which are employed in EQLM.

A. ϵ-Greedy Policies

To find an optimal solution, an agent must visit every state in the environment throughout its training, which requires the agent to explore the environment by periodically taking random actions. This conflicts with the agent's global goal of taking actions deemed optimal by the current control policy to improve the policy; thus the well known issue of balancing exploration and exploitation. One type of policies which help remedy this issue are known as "ϵ-greedy" policies [2]. In these policies, a parameter ϵ dictates the probability of taking a random action at each time-step, where $0 \leq \epsilon \leq 1$, and this can be tuned to give the desired trade-off between taking exploratory or exploitative actions. Exploration becomes less necessary later in the training period once the agent has more experience. Instead, the agent seeks to exploit actions considered more "optimal" following a period of more exploration. To achieve this in practice, ϵ varies linearly during the training period from ϵ_i to ϵ_f over N_ϵ episodes. Following this, ϵ is held constant at $\epsilon = \epsilon_f$ for the remainder of training. The exploration probability after n episodes is given by Equation 2.

$$\epsilon = \begin{cases} \epsilon_i - \frac{n}{N_\epsilon}\left(\epsilon_i - \epsilon_f\right), & \text{if } n < N_\epsilon \\ \epsilon_f, & \text{if } n \geq N_\epsilon \end{cases} \tag{2}$$

B. Target Network

A crucial issue with Q-networks is that they are inherently unstable and will tend to overestimate action-values, which can cause the predicted action-values to diverge [17]. Several methods to resolve this issue have been proposed, including the use of a target network [7]. This network calculates target action-values for updating the policy network and shares its structure with this network. The parameters of the policy network, θ are periodically transferred to the target network, whose parameters are denoted θ^-, which otherwise remain constant. In practise, the target network values are updated every C time-steps. This slightly decouples the target values from the policy network which reduces the risk of divergence.

C. Experience Replay

The online methods of learning discussed thus far all conventionally make updates on the most recent observed state transition, which has several limitations [7]. For example, states which are only visited once may contain useful update information which is quickly lost and updating on single state transitions results in low data efficiency of the agent's experience. A more data efficient method of performing updates utilises experience replay [18]. In this method, experiences of transitions are stored in a memory, $\mathcal{D}$ which contains the state s_j, action taken a_j, observed reward r_{j+1}, and observed state s_{j+1} for each transition. Updates are then made on a "minbatch" of k experiences selected randomly from the memory at every time-step. To limit the number of state transitions stored, a maximum memory size N_{mem} is defined such that a moving window of N_{mem} previous transitions are stored in the agent's memory [17].

D. Q-Network Algorithm

Figure 2 details the Q-Network algorithm which incorporates a target network and experience replay. This algorithm gives our baseline performance to which we compare Extreme

Fig. 2. Q-Network algorithm

```
1: initialise network with random weights
2: for episode= 1 to $N_{ep}$ do
3:     initialise state $s_t \leftarrow s_0$
4:     while state $s_t$ is non-terminal do
5:         select action $a_t$ according to policy $\pi$
6:         execute action $a_t$ and observe $r$, $s_{t+1}$
7:         update memory $\mathcal{D}$ with $(s_t, a_t, r_t, s_{t+1})$
8:         select random minibatch of k experiences $(s_j, a_j, r_j, s_{j+1})$ from $\mathcal{D}$
9:         $\mathbf{t}_j = \begin{cases} r_j, & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_a Q_T(s_{j+1}, a), & \text{otherwise} \end{cases}$
10:        $e_j = Q(s_j, a_j) - (r_{j+1} + \gamma \max_a Q_T(s_{j+1}, a))$
11:        update network using the error $e_j$ for each transition in the minibatch
12:        after $C$ time-steps set $\theta^- \leftarrow \theta$
13:    end while
14: end for
```

Q-Learning Machine (EQLM) and also provides the basis for incorporating ELM as a novel update mechanism.

III. ELM Theory and Development

A. Extreme Learning Machine

ELM in its most widely used form is a type of single-layer feedforward network (SLFN). The description of ELM herein uses the same notation as in [11]. Considering an arbitrary set of training data $(\mathbf{x}_i, \mathbf{t}_i)$ where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]$, a SLFN can be mathematically modelled as follows

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j, \;\; j = 1, \ldots, N \tag{3}$$

where $\tilde{N}$ is the number of hidden nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the output weight vector which connects the ith hidden node to the output nodes, $g(x)$ is the activation function, $w_i = [w_{i1}, w_{i2}, ..., w_{in}]^T$ is the input weight vector which connects the ith hidden node to the input nodes, and b_i is the bias of the ith hidden node. Where the network output $\mathbf{o}_j$ has zero error compared to the targets $\mathbf{t}_j$ for all N samples, $\sum_{j=1}^{\tilde{N}} \|\mathbf{o}_j - \mathbf{t}_j\| = 0$ it can be written that

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \;\; j = 1, \ldots, N \tag{4}$$

which contains the assumption that the SLFN can approximate the N samples with zero error. Writing this in a more compact form gives

$$\mathbf{H}\beta = \mathbf{T} \tag{5}$$

where $\mathbf{H}$ is the hidden layer output matrix, β is the output weight vector matrix, and $\mathbf{T}$ is the target matrix. These are defined as shown

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}} \tag{6}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m} \tag{7}$$

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \tag{8}$$

ELM performs network updates by solving the linear system defined in equation 5 for β

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T} \tag{9}$$

where $\mathbf{H}^\dagger$ here denotes the Moore-Penrose generalised inverse of $\mathbf{H}$ as defined in equation 10. This is used since, in general, $\mathbf{H}$ is not a square matrix and so cannot be inverted directly.

$$\mathbf{H}^\dagger = \left(\mathbf{H}\mathbf{H}^T\right)^\dagger \mathbf{H}^T \tag{10}$$

The method used by ELM to update its weights has several advantages over classical methods of updating neural networks. It is proven in [11] that $\hat{\beta}$ is the smallest norm least squares solution for β in the linear system defined by equation 5, which is not always the solution reached using classical methods. ELM also avoids many of the issues commonly associated with neural networks such as converging to local minima and improper learning rate. Such problems are usually avoided by using more sophisticated algorithms, whereas ELM is far simpler than most conventional algorithms.

B. Regularized ELM

Despite the many benefits of ELM, several issues with the algorithm are noted in [19]. Mainly, the algorithm still tends to overfit and is not robust to outliers in the input data. The authors propose a Regularized ELM which attempts to balance the empirical risk and structural risk to give better generalisation. This differs to the ELM algorithm which is solely based on empirical risk minimisation.

The main feature of regularized ELM is the introduction of a parameter $\bar{\gamma}$ which regulates the amount of empirical and structural risk. This parameter can be adjusted to balance the risks and obtain the best generalisation of the network. Weights are calculated as shown:

$$\beta = \left(\frac{I}{\bar{\gamma}} + \mathbf{H}^T D^2 \mathbf{H}\right)^\dagger \mathbf{H}^T \mathbf{T} \tag{11}$$

which incorporates the parameter $\bar{\gamma}$ and a weighting matrix D. Setting D as the identity matrix I yields an expression for unweighted regularized ELM:

$$\beta = \left(\frac{I}{\bar{\gamma}} + \mathbf{H}^T \mathbf{H}\right)^\dagger \mathbf{H}^T \mathbf{T} \tag{12}$$

which is a simplification of equation 11. ELM is then the case of equation 12 where $\bar{\gamma} \to \infty$. Adding the parameter $\bar{\gamma}$ adds some complexity to the ELM algorithm because of its tuning, however regularized ELM still maintains most of the advantages of ELM over conventional neural networks.

C. Incremental Extreme Learning Machine

It is desired to perform network updates sequentially on batches of data which necessitates an incremental form of ELM. Such an algorithm is presented in [20] whose basis is the regularized form of ELM shown in equation 12. The algorithm used for the purposes of EQLM is the least square incremental extreme learning machine (LS-IELM).

For an initial set of N training samples $(\mathbf{x}_i, \mathbf{t}_i)$ the LS-IELM algorithm initialises the network weights as shown:

$$\beta = A_t^\dagger \mathbf{H}^T \mathbf{T} \tag{13}$$

where

$$A_t = \frac{I}{\tilde{\gamma}} + \mathbf{H}^T \mathbf{H} \tag{14}$$

and $\mathbf{H}$ and $\mathbf{T}$ are given by equations 6 and 8. Suppose new sets of training data arrive in chunks of k samples - the hidden layer output matrix and targets for a new set of k samples are as shown:

$$\mathbf{H}_{IC} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_{N+k} + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_{N+k} + b_{\tilde{N}}) \end{bmatrix}_{k \times \tilde{N}} \tag{15}$$

$$\mathbf{T}_{IC} = \begin{bmatrix} \mathbf{t}_{N+1}^T \\ \vdots \\ \mathbf{t}_{N+k}^T \end{bmatrix}_{k \times m} \tag{16}$$

To perform updates using the most recent data at time t, K_t is defined as

$$K_t = I - A_t^\dagger \mathbf{H}_{IC}^T \left(\mathbf{H}_{IC} A_t^\dagger \mathbf{H}_{IC}^T + I_{k \times k} \right)^\dagger \mathbf{H}_{IC} \tag{17}$$

and the update rules for β and A are then as follows:

$$\beta_{t+1} = K_t \beta_t + K_t A_t^\dagger \mathbf{H}_{IC}^T \mathbf{T}_{IC} \tag{18}$$

$$A_{t+1}^\dagger = K_t A_t^\dagger \tag{19}$$

D. Extreme Q-Learning Machine

The algorithm for applying Q-learning using LS-IELM based updates, here referred to as the Extreme Q-Learning Machine (EQLM) is shown in Figure 3. Similar to the Q-network algorithm in Figure 2, this uses experience replay and a target network to improve its performance. Unlike the Q-network, the TD-error is not calculated and instead a target matrix, $\mathbf{T}$ for the minibatch of data is created which has the predicted action-values for all actions in the given states. The target action-value for each state, s_j is then assigned to the applicable value in $\mathbf{t}_j$. Matrix $\mathbf{H}$ is constructed using the states in the minibatch and then the update rules are applied. The boolean variable $step0$ is introduced to initialise the network at the very first update.

Fig. 3. EQLM algorithm

```
1:  initialise network with random weights
2:  $step0 \leftarrow True$
3:  for episode= 1 to $N_{ep}$ do
4:      initialise state $s_t \leftarrow s_0$
5:      while state $s_t$ is non-terminal do
6:          if episode$\leq N_h$ then
7:              select action $a_t$ according to heuristic $h_0(t)$
8:          else
9:              select action $a_t$ according to policy $\pi$
10:         end if
11:         execute action $a_t$ and observe $r$, $s_{t+1}$
12:         update memory $\mathcal{D}$ with $(s_t, a_t, r_t, s_{t+1})$
13:         select random minibatch of k experiences $(s_j, a_j, r_j, s_{j+1})$ from $\mathcal{D}$
14:         $\mathbf{t}_j = \begin{cases} r_j, & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_a Q(s_{j+1}, a), & \text{otherwise} \end{cases}$
15:         construct matrix $\mathbf{H}$
16:         if $step0$ then
17:             $A_t = \frac{I}{\tilde{\gamma}} + \mathbf{H}^T\mathbf{H}$
18:             $\beta_{t+1} = A_t^\dagger \mathbf{H}^T \mathbf{T}$
19:             $A_{t+1} = A_t$
20:             $step0 \leftarrow False$
21:         else
22:             $K_t = I - A_t^\dagger \mathbf{H}^T \left( \mathbf{H} A_t^\dagger \mathbf{H}^T + I_{k \times k} \right)^\dagger \mathbf{H}$
23:             $\beta_{t+1} = K_t \beta_t + K_t A_t^\dagger \mathbf{H}^T \mathbf{T}$
24:             $A_{t+1}^\dagger = K_t A_t^\dagger$
25:         end if
26:         after $C$ time-steps set $\theta^- \leftarrow \theta$
27:     end while
28: end for
```

One further key difference in the EQLM algorithm is the heuristic policy used in initial episodes. The return in initial episodes has a substantial effect on the convergence of EQLM as discussed later. This necessitates a simple heuristic controller for the start of training which does not need to perform very well, but can at least prevent the agent from converging on a highly sub-optimal policy. EQLM uses a heuristic action selection $a_t = h_0(t)$, which is effectively an open loop control scheme dependant only on the time-step, for N_h episodes. Definition of this heuristic is discussed in the following section.

IV. Experiments and Results

Code to reproduce results is available at https://github.com/strath-ace/smart-ml.

A. OpenAI Gym Environments

The environment used to test the algorithms comes from the OpenAI Gym which is a toolkit containing benchmark tests for a variety of machine learning algorithms [21]. The gym contains, among other environments, several classical control problems, control tasks which use the MuJoCo physics engine

[22], and the Atari2600 games which are used in [7]. Here the agents will be tested on the environment named “CartPole-v0”.

The cart-pole problem was originally devised in [23] where the authors created an algorithm called “BOXES” to learn to control the system. In this task, a pole is attached by a hinge to a cart which rolls along a track and is controlled by two possible actions - an applied force of fixed magnitude in either the positive or negative x-direction along the track. The goal is to keep the pendulum from toppling for as long as possible, which yields a very simple reward function of $r = +1$ for every time-step where the pendulum has not toppled. In addition, the track on which the cart is situated is finite and reaching the limits of the track also indicates failure. The dynamics of the system used in the gym are the same as those defined by [24]. The state-space size for this environment is 4 and the action-space size is 2.

This problem can be considered an “episodic” task [2], where the learning is divided into episodes which have defined ending criteria. An episode terminates either when the pendulum passes 12° or the cart reaches either end of the track. In this task, a maximum number of time-steps per episode of 200 is set within the gym.

B. Heuristic Policy

As discussed previously, EQLM is susceptible to converging on a sub-optimal policy without the use of a heuristic policy in the initial episodes. A random policy at the start of training will sometimes produce this sub-optimal result and so we need to define a simple deterministic policy which does not immediately solve the task but prevents unacceptable long-term performance. For the cart-pole task we consider here which has a binary action space, we define the heuristic policy as taking alternating actions at each time-step as shown:

$$h_0(t) = mod(t, 2) \tag{20}$$

From testing, we found $N_h = 5$ to be a suitable number of episodes over which to use the heuristic. The effect of this initial heuristic policy is shown in Figure 4. This shows the averaged rewards over the first 200 episodes of training for both networks with and without the heuristic. While the return in the initial episodes is still higher for EQLM in both cases, it is clear that with the heuristic EQLM shows a more favourable performance. This is due to occasions where, without the heuristic, EQLM quickly converges to a sub-optimal policy, which is mitigated by the heuristic policy. Also shown is the average performance of the heuristic alone, which receives a reward of 37 per episode. This indicates that although the heuristic alone performs very poorly on the task, it is still useful to improve the performance of both algorithms.

C. Hyperparameter Selection

The performance of a Q-learning agent can be very sensitive to its hyperparameters. To create a useful comparison of each agent’s performance we therefore need to tune the hyperparameters for this problem. Here we use the Python library Hyperopt which is suitable for optimising within combined

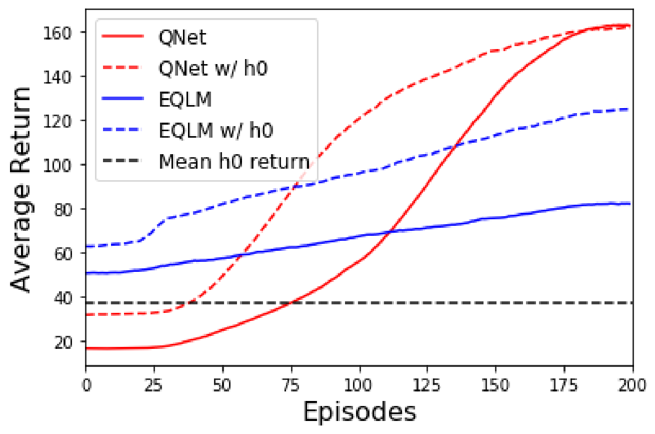

Fig. 4. Varying performance with the use of an initial heuristic h_0 with the average performance for the heuristic alone shown

discrete- and real-valued search spaces [25]. Hyperparameters to be optimised are: learning rate α (Q-Network only), regularisation parameter $\bar{\gamma}$ (EQLM only), number of hidden nodes $\tilde{N}$, initial exploration probability ϵ_i (with ϵ_f fixed as 0), number of episodes to decrease exploration probability N_ϵ, discount factor γ, minibatch size k, and target network update steps C.

Our main objective to optimise is the final performance of the agent, i.e. the total reward per episode, after it converges to a solution. In addition, an agent should converge to the optimal solution in as few episodes as possible. Both these objectives can be combined into the single metric of area under the learning curve as shown in Figure 5. Since hyperopt uses a minimisation procedure, we specifically take the negative area under the curve. One of the issues with optimising these systems is their stochastic nature which can result in several runs with the same hyperparameters having vastly different performance. To account for this, each evaluation uses 8 runs and the loss is the upper 95% confidence interval of the metric from these runs. This gives a conservative estimate of the worst-case performance for a set of hyperparameters.

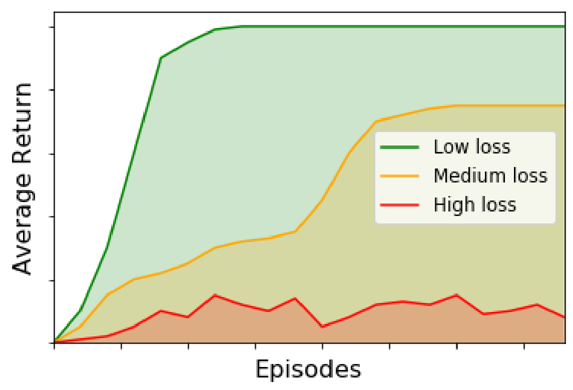

Fig. 5. Example learning curves which show different values for the loss function

Table I shows the best parameters obtained when tuning the hyperparameters for this task. Most of the hyperparameters common to each algorithm are not substantially different with the exception of minibatch size, k which is 26 and 2 for the Q-network and EQLM respectively. In fact, the performance of EQLM tended to decrease for larger values of k which was not the case for the Q-network. This could be a result of the matrix inversion in EQLM where the behaviour of the network is less stable if the matrix is non-square. Alternatively, it is possible that EQLM attempting to fit to a much larger number of predicted Q-values causes the overall performance to decrease. The fact it needs fewer data per time-step than a standard Q-network could also indicate that EQLM is more efficient at extracting information on the environment's action-values compared to using gradient descent.

Hyperparameter	Q-Network	EQLM
α	0.0065	-
$\tilde{\gamma}$	-	1.827e-5
$\tilde{N}$	29	25
ϵ_i	0.670	0.559
N_ϵ	400	360
γ	0.99	0.93
k	26	2
C	70	48

TABLE I
HYPERPARAMETERS USED FOR EACH AGENT IN THE CART-POLE TASK

D. *Learning Performance*

With the selected hyperparameters, each agent carried out 50 runs of 600 episodes in the cart-pole environment to compare their performance. The results of this are shown in Figure 6 and Table II. Here we use two measures of performance: mean reward over the final 100 episodes and area under the learning curve (auc).

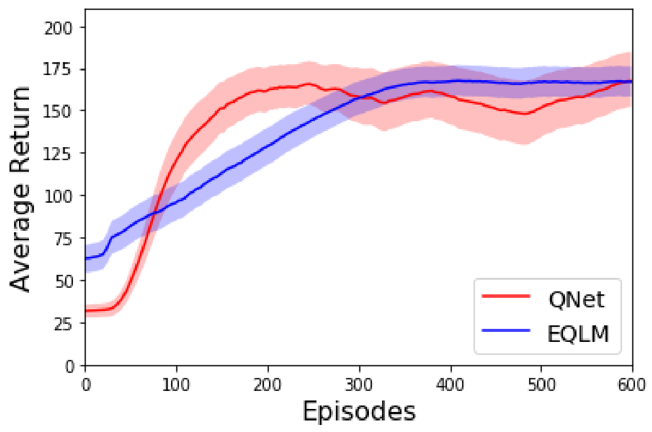

Fig. 6. Learning curves for EQLM and a standard Q-Network in the cart-pole task. Results are averaged over all 50 runs at each episode and shaded area indicates the 95% confidence interval

From the learning curves, we see EQLM on average achieves a superior performance in the earliest episodes of training followed by a steady increase in return until it plateaus. The Q-network begins with comparatively low average return but then shows a sharp increase in return before its performance plateaus for the remainder of the episodes. After each of the learning curves plateau at their near-optimal performance, we see some of the most interesting differences between the two algorithms. The average return for EQLM remains very consistent as do the confidence intervals, however the Q-network displays some temporal variation in its performance as training continues and the confidence intervals tend to get larger. This shows that the long-term performance of EQLM is more consistent than the equivalent Q-network, which is backed up by the data in Table II. The standard deviation of the mean reward of EQLM (23.1) is less than half that of the Q-network (47.0) and both algorithms have comparable mean rewards (160.0 and 166.9 for Q-network and EQLM respectively).

Measure		Q-Network	EQLM
reward	mean	160.0 (147.5, 173.7)	166.9 (160.7, 173.3)
	std	47.0 (35.1, 62.2)	23.1 (20.3, 26.7)
auc ($*10^3$)	mean	84.1 (81.0 87.4)	83.3 (80.4, 86.2)
	std	11.7 (9.1, 14.7)	10.6 (9.3, 12.4)

TABLE II
PERFORMANCE OF EACH ALGORITHM IN THE CART-POLE TASK

To find a statistical measure of the difference in performance of each algorithm, we use a two-tailed t-test [26]. This assumes both algorithms' performance belongs to the same distribution which we reject when the p-value is less than a threshold of 0.05. When comparing the mean reward in the final episodes, the t-test yielded values of $t = -0.628$, $p = 0.531$. Similarly for the area under the learning curve we obtained $t = -1.16$, $p = 0.24$. As a result, we cannot reject the hypothesis that the performance of both algorithms follows the same distribution. This demonstrates EQLM as being capable of achieving similar performance to a standard Q-Network in this task.

V. CONCLUSION

This paper proposed a new method of updating Q-networks using techniques derived from ELM called Extreme Q-Learning Machine (EQLM). When compared to a standard Q-network on the benchmark cart-pole task, EQLM shows comparable average performance which it achieves more consistently than the Q-network. EQLM also shows better initial learning performance when initialised using a basic heuristic policy.

While EQLM shows several advantages to standard Q-networks, it is clear that the conventional gradient descent methods are also capable of learning quickly as they gain more experience. Future work could look at combining the strengths of EQLM's initial performance and using gradient-based methods to accelerate the learning. In this paper we have tuned the hyperparameters of EQLM for a specific problem, but a more rigorous parametric study is necessary to learn more about the effect of the hyperparameters on EQLM's learning performance. One of the developments in

ELM which was not used here is the ELM-based multilayer perceptron [27]. Such a network could similarly be used for RL problems since deep networks are generally better suited to more complex tasks [28].

The results in this paper suggest ELM methods are capable of being used within RL with similar performance and greater consistency than conventional gradient-descent for simple RL problems. Additional research is needed on the application of EQLM to higher dimensional and adaptive control problems.

Acknowledgment

The authors would like to thank the University of Strathclyde Alumni Fund for their support.

References

[1] R. E. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, p. 503–516, 1954.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998.

[3] R. S. Sutton, "Generalization in Reinforcement Learning : Successful Examples Using Sparse Coarse Coding," *Advances in Neural Information Processing Systems*, vol. 8, pp. 1038–1044, 1996.

[4] R. Davoodi and B. J. Andrews, "Computer simulation of FES standing up in paraplegia: A self-adaptive fuzzy controller with reinforcement learning," *IEEE Transactions on Rehabilitation Engineering*, 1998.

[5] 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.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, 2016.

[7] 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, 2015.

[8] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International conference on Machine Learning*, 2016.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning With Double Q-Learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[10] D. E. Moriarty and A. C. Schultz, "Evolutionary Algorithms for Reinforcement Learning," *Journal of Artiicial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[11] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[12] G. B. Huang, D. H. Wang, and Y. Lan, "Extreme Learning Machines: A Survey," *International Journal of Machine Learning and Cybernetics*, vol. 2, no. 2, pp. 107–122, 2011.

[13] J. M. Lopez-Guede, B. Fernandez-Gauna, and M. Graña, "State-Action Value Function Modeled by ELM in Reinforcement Learning for Hose Control Problems," *International Journal of Uncertainty*, vol. 21, pp. 99–116, 2013.

[14] J. M. Lopez-Guede, B. Fernandez-Gauna, and J. A. Ramos-Hernanz, "A L-MCRS dynamics approximation by ELM for Reinforcement Learning," *Neurocomputing*, vol. 150, pp. 116–123, 2014.

[15] T. Sun, B. He, and R. Nian, "Target Following for an Autonomous Underwater Vehicle Using Regularized ELM-based Reinforcement Learning," in *OCEANS'15 MTS/IEEE Washington*, 2015, pp. 1–5.

[16] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, 1989.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[18] L.-J. Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992.

[19] W. Deng, Q. Zheng, and L. Chen, "Regularized Extreme Learning Machine," *IEEE Symposium on Computational Intelligence and Data Mining*, no. 60825202, pp. 389–395, 2009.

[20] L. Guo, J. h. Hao, and M. Liu, "An incremental extreme learning machine for online sequential learning problems," *Neurocomputing*, vol. 128, pp. 50–58, 2014.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," pp. 1–4, 2016.

[22] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

[23] D. Michie and R. A. Chambers, "BOXES: An Experiment in Adaptive Control," *Machine Intelligence*, vol. 2, pp. 137–152, 1968.

[24] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[25] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *30th International Conference on Machine Learning, ICML 2013*, vol. 28, no. PART 1, 2013, pp. 115–123.

[26] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," in *The Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 3207–3214.

[27] J. Tang, C. Deng, and G.-B. Guang, "Extreme learning machine for multilayer perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 4, pp. 809–821, 2015.

[28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.