Deep W-Networks: Solving Multi-Objective Optimisation Problems With Deep Reinforcement Learning

Jernej Hribar, Luke Hackett, and Ivana Dusparic
School of Computer Science and Statistics, Trinity College Dublin, Ireland
{jhribar, lhackett, Ivana.Dusparic}@tcd.ie

Keywords: Deep Reinforcement Learning, Deep Q-Networks, W-Learning, Deep W-Networks, and Multi-objective.

Abstract: In this paper, we build on advances introduced by the Deep Q-Networks (DQN) approach to extend the multi-objective tabular Reinforcement Learning (RL) algorithm W-learning to large state spaces. W-learning algorithm can naturally solve the competition between multiple single policies in multi-objective environments. However, the tabular version does not scale well to environments with large state spaces. To address this issue, we replace underlying Q-tables with DQN, and propose an addition of W-Networks, as a replacement for tabular weights (W) representations. We evaluate the resulting Deep W-Networks (DWN) approach in two widely-accepted multi-objective RL benchmarks: deep sea treasure and multi-objective mountain car. We show that DWN solves the competition between multiple policies while outperforming the baseline in the form of a DQN solution. Additionally, we demonstrate that the proposed algorithm can find the Pareto front in both tested environments.

1 Introduction

Many real world problems such as radio resource management (Giupponi et al., 2005), infectious disease control (Wan et al., 2020), energy-balancing in sensor networks (Hribar et al., 2022), etc., can be formulated as a multi-objective optimisation problem. Whenever an agent is tackling such a problem in a dynamic environment, a single objective Reinforcement Learning (RL) methods such as Q-learning will not result in a behaviour that will be optimal for all objectives. Instead, the single objective solution will most likely prefer one objective over others. Alternatively, the agent can employ a Multi-Objective Reinforcement Learning (MORL) method. Unfortunately, while much work has been completed in tabular MORL (Liu et al., 2015), the curse of dimensionality limits the applicability of these methods to real-world problems. The curse of dimensionality refers to the challenges that come with organizing and analyzing data that has an intractably and/or infinitely large state-space, for example, using images as input states. Recent developments in RL merged Q-Learning with Neural Networks (Mnih et al., 2013), vastly expanding the complexity of problems that could be tackled with RL. Work has been done in the past few years in order to employ Deep Q-Networkss (DQNs) to solve Multi-Objective problems (Liu et al., 2015). However, most of the proposed solutions have drawbacks. These drawbacks range from a required high number of sampled experiences to train the neural networks, which will take an extended amount of time, to adding complexity by creating new types of networks with altered memory storage. To overcome these obstacles, in this paper, we propose a deep learning extension to a tabular multi-objective technique called W-learning (Humphrys, 1995).

W-Learning was first proposed in the late 90's (Humphrys, 1995), as a multi-policy way to solve multi-Objective problems that use Q-Learning agents with a single objective as part of a larger system. The main principle is that there will be many Q-Learning agents, each with a different policy. These agents will all suggest an action that will be selfish, and the best action will need to be determined from these suggested selfish actions. The goal of W-Learning is to determine which of these actions should be selected, ensuring the right agent "wins". The way it determines this is by attempting to figure out how much the agent cares about the action that it suggests. Some scenarios or states might not impact the reward of certain agents so much however it may massively impact one or some of them. W-learning has been successfully applied in a range multi-objective problems, from speed limit control on highways (Kusic et al., 2021), smart grid (Dusparic et al., 2015), and conflict detection in self-adaptive systems (Cardozo and Dusparic, 2020). However, as W-learning was proposed long before deep learning was successfully implemented in RL to deal with the curse of dimensionality (Mnih et al., 2013), this limited the domains and scale of problems that W-Learning can be applied to. In this paper, we take advantage of advances introduced by DQN to train multiple Artificial Neural Network (ANN) on different objectives and propose a new "deep" variant of W-Learning to address competition between these objectives.

One of the most significant advantages of multi-policy algorithms over single-policy, e.g., Q-learning, is in their ability to find the *Pareto Optimal* behaviour(Jin and Sendhoff, 2008). To be Pareto Optimal, the agent's actions must be such that an improvement in its decision process for an objective will not harm the reward for any other objective. Single-policy algorithms rely on some specification of preferences for given objectives and, therefore, will not necessarily find the policy that will result in the optimal Pareto front. In other words, multi-objective algorithms require less information about the environment before training and are generally more favoured for use in offline learning.

Our proposed Deep W-Networks (DWN) algorithm takes advantage of the computational efficiency of single-policy algorithms by considering each objective separately. These policies will suggest a selfish action that will only maximise their own reward. However, the DWN resolves the competition between greedy single-objective policies by relying on W-values representing policies' value to the system. These W-values can be learned with interaction with the environment, following logical steps similar to a well-known Q-learning algorithm. In our proposed implementation, we employ two DQNs for each objective. One DQN is used to learn a greedy policy for the given objective, while the second DQN has only one output representing the policy's W-value for a given state input. Furthermore, DWN has the benefit of training all policies simultaneously, which allows for a faster learning process. Additionally, DWN can take advantage of modularity, meaning that policies can be trained separately and then included in the DWN agent. Modularity also enables policies to be altered, e.g., the reward function is changed, added, or deactivated, without the need to re-train all other policies.

The rest of the paper is organised as follows. In the next section, we discuss the most important design features of Deep Reinforcement Learning (DRL) introduced in the last decade and related work. In section 3 we present and describe our proposed DWN algorithm. Followed by evaluation section 4, in which we employ two multi-objective environments: multi-objective mountain car and deep sea treasure. We show in both environments that DWN is capable of resolving the competition between multiple policies while outperforming the baseline in the form of DQN solution. Finally, we provide concluding remarks in section 5.

2 Background

In this section, we introduce the essential elements required to understand DRL and review related algorithms capable of resolving multi-objective problems.

2.1 Deep Reinforcement Learning

The goal of RL algorithm is to find the optimal policy π_* for an environment that is fully characterised with an Markov Decision Process (MDP) (LeCun et al., 2015). With MDP we describe a sequential decision-making process in a form of a state-space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function R, and a set of transition probabilities between the states $\mathcal{P}$. In such settings, the optimal policy π_* is the policy that will maximise the long-term reward.

An example of a DRL algorithm that revolutionised the field in 2013 is DQN. The DQN algorithm is a deep learning extension of a well-known action-value algorithm named Q-learning. In action-value based methods, the agent interacts with the environment by taking actions a and receiving a reward r that indicates if the taken action was desirable or not. The Q represents the quality of an action-value $Q(s,a)$, with s representing the state. The objective of the RL algorithm is to accurately estimate Q values for all action-values using a Bellman equation. Once the agent can accurately determine all values, it can find the optimal policy π_* for selection actions that will maximise the expected reward $r+\gamma Q(s',a')$, with γ representing the discount for rewards obtained in next time-step. The Q-values are updated in iterations as follows:

$$Q_{i+1}(s,a) = \mathbb{E}_{s\sim\mathcal{S}}\left[r = \gamma \max Q^*(s',a')|s,a\right]; \quad (1)$$

and converge to the optimal value when:

$$Q_i \rightarrow Q^* \quad \text{as} \quad i \rightarrow \infty. \quad (2)$$

However, such an iterative approach requires the agent to explore the entire state space, i.e., try all possible action in every state. In practice, such an approach is impossible as such exploration would take a gargantuan amount of time and computational power. Instead, a state-space approximator is used. An example of a very effective non-linear state-space approximator is ANN.

In the DQN, a ANN function approximator with weights θ is employed to represent Q-network. The agent uses the network to estimate the action-values, i.e., $Q(s,a;\theta) \approx Q^*(s,a)$. The values in the ANN are updated, i.e., trained, at each iteration i by minimising the loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim S}\left[(y_i - Q(s,a;\theta_i))^2\right], \quad (3)$$

where $y_i = \mathbb{E}_{s\sim S}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}|s,a]$ represent target and ρ is the probability distribution over sequences. Note that when determining the target, the values from the previous iteration, i.e., θ_{i-1}, are held fixed. The gradient of the loss function is then determined as:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim S}\big[\big(r+\gamma\max_{a'} Q(s',a';\theta_{i-1} - Q(s,a)\big)\nabla_{\theta_i} Q(s,a;Q_i)\big]. \quad (4)$$

In practice, the loss function is optimised using gradient descent as it is less computation-intensive than computing the expectation $\mathbb{E}$ directly. However, the training process can be unstable and prone to converge to a local optimum. To remedy this issue, the DQN introduced experience replay and the use of policy and target ANN.

Experience replay is a batch memory $\mathcal{M}$ into which the agent is storing experiences. An experience typically consist of a state, the next state, selected action, and obtained reward, i.e., a tuple $< s,a,s',r >$. The agent often samples experiences uniformly randomly during the training process. However, because not all experiences are equal in terms of importance to the learning process, a much better approach is to prioritise them, i.e., increase the probability of their selection. Using Prioritised Experience Replay (PER)(Schaul et al., 2015) we determine the probability of sampling an experience i as:

$$P(i) = \frac{p_i^\zeta}{\sum_K p_K^\zeta}, \quad (5)$$

with factor $\zeta, \zeta \in [0,1]$ controlling the degree to which experiences are prioritised. Using PER can significantly reduce the time the agent requires to find the optimal policy. To further stabilise the learning process, the DQN algorithm introduced the use of target ANN for estimating the $Q(s',a')$ during training. Such an approach is necessary because using the same ANN for determining both Q-values can results in very similar estimations due to possible small difference in s and s'. Therefore, using a target policy for $Q(s',a')$ estimation can prevent such occurrences.

In our work, we adopt these aforementioned advances to extend the original W-learning algorithm usability in large state spaces.

2.2 Multi-Objective Reinforcement Learning

Existing RL methods employed for resolving multi-objective optimisation problems can be generalised into two groups: tabular RL and DRL methods.

An example of a tabular methods are GM-Sarsa(0) (Sprague and Ballard, 2003) and its extension with weighted sum approach (Karlsson, 1997). GM-Sarsa(0) (Sprague and Ballard, 2003) aims to find good policies concerning the composite tasks as opposed to finding the best policy for each task and then merging these into a single policy. On the other hand, the authors in (Karlsson, 1997) proposed to use a synthetic objective function to emphasise the importance of each objective in the form of weight for Q-values. Unfortunately, neither of these methods performs well in finding the optimal multi-objective policy. Additionally, the performance of the tabular methods deteriorates when applied in environments with large state spaces.

The second group, DRL methods (Mossalam et al., 2016; Nguyen et al., 2020; Tajmajer, 2018; Abels et al., 2019), can deal with large state spaces. The deep optimistic linear support learning (Mossalam et al., 2016) is an example of the first known extension of the DQN that dealt with multi-objectivity. The limitation of linear support approach is in redundant computations and additional required representations. Both limitations can be overcome in two-stage multi-objective DRL (Nguyen et al., 2020) approach. In the latter, once policies are learned, policy-independent parameters are tuned using a separate algorithm that attempts to estimate the Pareto frontier of the system. Similarly, modular multi-objective DRL with subsumption architecture (Tajmajer, 2018) was proposed that combines the results of single policies, represented by a DQN, to take the action most amenable to all rewards for each environment step. The approach resembles a voting system with Q-values representing a vote for a certain policy. Finally, dynamic weights in multi-objective DRL (Abels et al., 2019) were proposed to deal with situations where the relative importance of weights changes during training. In contrast, our proposed DWN has the benefit of simultaneously training all policies, which allows for a faster dynamic adjustment of policy rewards that the above DRLs methods lack.

3 Deep W-Learning Framework

In this section, we detail out our proposed multi-objective DWN approach. We denote the multi-objective environment with tuple $< N, \mathcal{S}, \mathcal{A}, \mathcal{R}, \Pi >$ in which N is the number of policies, $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the set of all available actions, $\mathcal{R} = \{R_1, ..., R_N\}$ is set of reward functions, and $\Pi = \pi, ..., \pi_N$ denotes the set of available policies. At each decision epoch t policies observe the same state $s(t) \in \mathcal{S}$ and every policy suggest an action. The agent's objective is to select the best possible action from the vector of actions nominated by agents for execution at that time-step $\mathbf{a}(t) = \{a_1(t), ..., a_N(t)\}$. Furthermore, we denote the selected action at time-step t with index j, i.e., the selected action is denoted as $a_j(t)$. Additionally, within our environment, we use Π_{-j} to denote the set containing all polices except the j-th one.

The agent making the decision in a multi-objective environment at each time-step has to determine to what extent it will take into account each of the objectives. In other words, the agent has to continuously keep resolving the competition between multiple objectives. In W-learning, each objective is represented with a single Q-learning policy; each Q-learning policy has a different goal and, depending on the observed state $s(t)$, suggests a greedy action $a_i(t)$. These actions are often conflicting with each other, and to resolve the competition, the agent learns a table of weights for each state, called the W-values. For each observed state $s(t)$, the agent obtains $W_i(t)$ where i is the index of the policy. The agent then takes the action suggested by the policy associated with the highest $W_i(t)$:

$$W_j(t) = \max\left(\{W_1(t), ..., W_N(t)\}\right). \tag{6}$$

Note that the index j marks the W-value of the policy the agent has selected. Updating the W-values follows a very similar formulation as updating the Q-values(Eq.1):

$$W_i(t) \leftarrow (1-\alpha)W_i(t) + \alpha\big[Q(s(t), a_j(t)) - (R_i(t) + \gamma \max_{a_i(t+1)\in\mathcal{A}} Q(s(t+1), a_i(t+1))\big]. \tag{7}$$

However, the agent will not update the W-value of the selected policy, i.e., the agent only updates W-values for set of policies Π_{-j}. Excluding W-Learning update for the selected policy allows other policies to emerge as the leader overtime. Such an approach is acceptable in practice as polices become more adept at their given task. Additionally, the updating constraint only applies for W-values, the agent will update Q-values for every policy, i.e., for the set Π.

Algorithm 1 Deep W-Learning with PER

```
 1: Input: Minibatch size $K$, replay memory size $M$, exploration rates $\varepsilon^Q$, $\varepsilon^W$, smoothing factor $\zeta$, exponent $\beta$, w-learning learning rate $\alpha$, soft update factors $\tau^Q$, $\tau^W$
 2: Initialize $N$ Q-networks $\theta_i^Q$, target $\hat{\theta}_i^Q$, and replay memory $\mathcal{D}_i^Q$ $\forall i \in \{1, \ldots, N\}$
 3: Initialize $N$ W-networks $\theta_i^W$, target $\hat{\theta}_i^W$, and replay memory $\mathcal{D}_i^W$ $\forall i \in \{1, \ldots, N\}$
 4: for $t = 0$ to $T-1$ do
 5:     Observe $s(t), s(t) \in \mathcal{S}$
 6:     // Nominate actions using epsilon-greedy
          approach ($\varepsilon_Q$)
 7:     Get $\mathbf{a}(t) = \{a_1(t), ..., a_N(t)\}$
 8:     Get $W_i(t) \forall i \in \{1, \ldots, N\}$
 9:     // Select and execute action using
          epsilon-greedy approach ($\varepsilon_W$)
10:     Get $j$-th policy with the highest W-value
          using Eq. 6
11:     Execute action $a_j(t)$, and observe
          state $s(t+1)$
12:     Initiate policy training with Alg. 2
13:     Initiate W training with Alg. 3
14: end for
```

In our proposed DWN implementation, each policy has two DQN networks. The role of the first DQN is to determine Q-values for the given policy, i.e., for greedy actions the policy will suggest. On the other hand, the second DQN has only one output, representing the W-value, and replaces a tabular representation of state-W-value pairs present in the original W-learning implementation. We summarise the proposed DWN in Alg. 1. Each policy requires two replay memories, one for Q-networks and another for the W-networks. Such a split is necessary because the agent will store a W-learning experience only when the agent did not select the policy. Additionally, we employed PER (Schaul et al., 2015) in our implementation to expedite the learning process.

The most significant aspect of the proposed DWN algorithm is the action-nomination step. In the action nomination step, each policy suggests a greedy action with an epsilon probability that it will select a random action. Similarly, the agent will select the policy with the highest W-value but with epsilon probability,

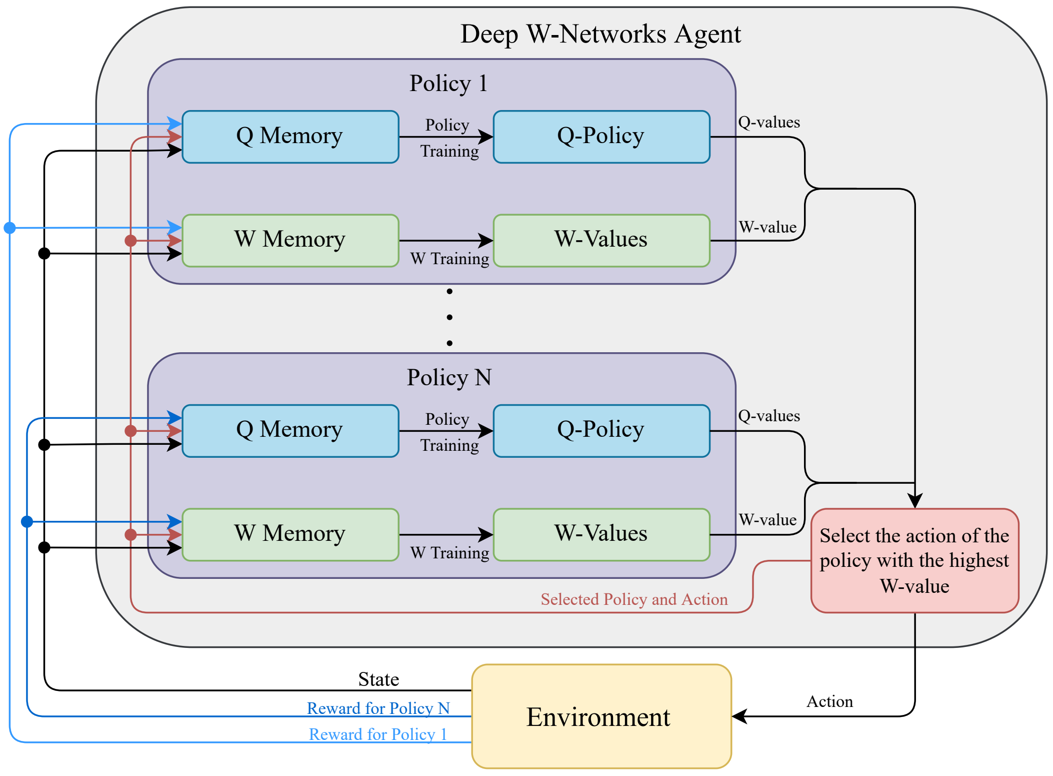

Figure 1: Deep W-Networks architecture.

Algorithm 2 Policy Training

```
1:  Input:$\Pi$, $\mathcal{D}_i^Q, i \in \{1,\ldots,N\}$, $K$, $M$, $\zeta$, $\beta$, $\theta_i^Q, i \in \{1,\ldots,N\}$, $\hat{\theta}_i^Q, i \in \{1,\ldots,N\}$
2:  for each $\pi_i$ in $\Pi$ do
3:      Determine reward $r_i(t)$ using function $R_i$
4:      Store experience $(s(t), a_j(t), r_i(t), s(t+1)$
        in $\mathcal{D}_i^Q$ with priority $p_t = \max_{m<t} p_m$
5:      for each $k = 1$ to $K$ do
6:          Sample transition $k \sim P(k) = p_k^\zeta / \sum_m p_m^\zeta$
7:          Compute importance-sampling weight:
            $\omega_k = (MP(j))^{-\beta} / \max_m \omega_m$
8:          Update transition priority $p_k \leftarrow |\delta_k|$
9:          Accumulate weight-change:
            $\Delta \leftarrow \Delta + \omega_k \cdot \delta_k \cdot \nabla_{\theta_i} Q(s(k-1), a_j(k-1)$
10:     end for
11:     Update weights $\theta_i^Q \leftarrow \theta_i^Q + \eta \cdot \Delta$, reset $\Delta = 0$
12:     Soft update target network:
            $\hat{\theta}_i^Q \leftarrow \tau^Q \theta_i^Q + (1-\tau^Q)\hat{\theta}_i^Q$
13:     Update $\varepsilon^Q$ using decay
14: end for
```

Algorithm 3 W Training

```
1:  Input:$\Pi$, $\mathcal{D}_i^W, i \in \{1,\ldots,N\}$, $K$, $M$, $\zeta$, $\beta$, $\theta_i^W, i \in \{1,\ldots,N\}$, $\hat{\theta}_i^W, i \in \{1,\ldots,N\}$
2:  for each $\pi_i$ in $\Pi_{-j}$ do
3:      Determine reward $r_i(t)$ using function $R_i$
4:      Store experience $(s(t), a_j(t), r_i(t), s(t+1)$
        in $\mathcal{D}_i^W$ with priority $p_t = \max_{m<t} p_m$
5:  end for
6:  for each $\pi_i$ in $\Pi$ do
7:      for each $k = 1$ to $K$ do
8:          Sample transition $k \sim P(k) = p_k^\zeta / \sum_m p_m^\zeta$
9:          Compute importance-sampling weight:
                $\omega_k = (MP(j))^{-\beta} / \max_m \omega_m$
10:         Update transition priority $p_k \leftarrow |\delta_k|$
11:         Accumulate weight-change $\Delta$ using Eq. 7
12:     end for
13:     Update weights $\theta_i^W \leftarrow \theta_i^W + \eta \cdot \Delta$, reset $\Delta = 0$
14:     Soft update target network:
            $\hat{\theta}_i^W \leftarrow \tau^W \theta_i^W + (1-\tau^W)\hat{\theta}_i^W$
15:     Update $\varepsilon^W$ using decay
16: end for
```

it might decide to explore, i.e., select the policy randomly. W-values exploration is necessary to avoid a single policy prevailing at the start of the learning due to high randomly initialized values and batch learning. Before the learning process can start, the agent requires a minimum of K experiences stored in the replay memory to start the training process.

We summarise the steps to train the policy and W-networks in two algorithms and give an overview of the DWN architecture in Fig. 1. During policy training (Alg. 2), each policy network optimises for the highest reward for its target. Note that this process remains unchanged from the original DQN implementation, but is an integral part of the proposed DWN. After Q networks have been through a few updates the W training (Alg. 3) can begin. We achieve the delay by keeping the batch size for both policy and W training the same, or greater. The W policy saves the experience only when it was not selected. In Alg. 3, line 3 we save the W experiences of all policies but j-th, which was selected (in line 10, Alg. 1). Consequently, we achieve the delay in training the W networks. Epsilon greedy approach of selecting W-values ensures that the agent does not select the same W network in every step at the start of the training. In the next section, we demonstrate how DWN performs in a multi-objective environment.

4 Evaluation

In this section, we evaluate[1] the proposed DWN using two multi-objective environments: multi-objective mountain car and deep sea treasure. The state space in the first environment is hand-crafted and is represented by only a two-input state vector. The two inputs are the car's position and velocity. The simplified case enables us to analyse DWN performance in more detail. On the other hand, in the second environment, the deep sea treasure, we use visual inputs as states to demonstrate that DWN performs well in environments with large state spaces.

4.1 Multi-Objective Mountain Car

The first environment, called *Multi-Objective Mountain Car* presents a scenario where a car is stuck in the middle of a valley. The car must reach the top of the valley. However, the car does not have enough power to reach the top by driving directly forward. Instead, the agent has to learn to first move away from its objective, by reversing up the hill to gain momentum, in order to reach it. We use the environment, with minor modifications, as defined in (Vamplew et al., 2011). The only alteration we made is the maximal number of steps allowed in an episode. We set the limit to 2000 because the goal of analysis in this environment is to gain a deeper understanding of DWN performance.

The environment has three different objectives: time penalty, backward acceleration penalty, and forward acceleration penalty. As the name of each policy suggests, the time policy gives a negative reward in every time step, except in the state when the agent reaches the top, the backward acceleration policy gives a negative reward when the agent is accelerating backward, and analogously, forward acceleration penalty applies for the forward policy. The agent has three available actions: accelerate forwards, accelerate backward, or do nothing. We design the DWN agent with three policies, one for each objective. For simplicity, we use the same ANN structure for all policies and for both Q and W networks, i.e., $\theta_i^Q, \hat{\theta}_i^Q, \theta_i^W, \hat{\theta}_i^W \forall i$. We use a feedforward ANN structure with two hidden layers, each with 128 neurons. On the output layer, to ensure better stability of learning, we employ a dueling network architecture (Wang et al., 2016), with 256 neurons. We list hyper-parameters in Table 1.

In Fig. 2 we show the average number of steps, averaged over ten episodes, the agent requires to reach the top of the hill. The proposed DWN and DQN algorithms both achieve similar performance in the same number of episodes. Note that the DQN receives the reward in the form of a sum of three reward signals, one for each policy the environment has. Furthermore, for a fair comparison DQN employ the same ANN structure as our DWN. Overall, the DWN performance is similar, albeit slightly more stable, to DQN in the mountain car environment. A far more interesting analysis is in individual policy performance

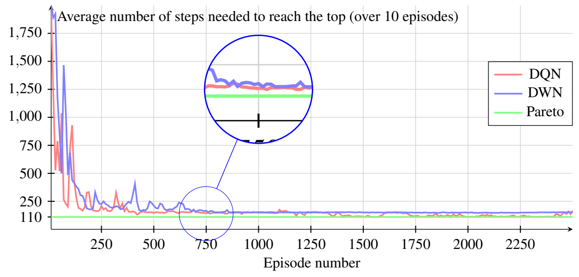

Figure 2: The number of steps, averaged over 10 episodes, each approach requires to finish the episode, i.e., the car reaching the top of the hill.

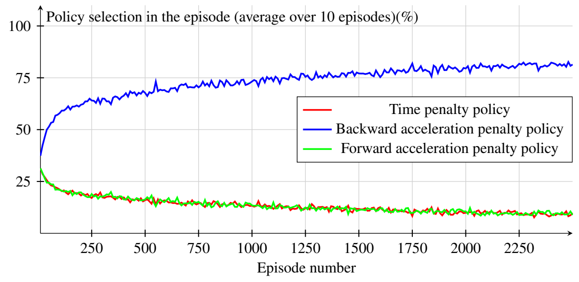

Figure 3: The percentage each policy in DWN agent selects in an episode, averaged over 10 episodes.

[1]DWN algorithm implementation and evaluation code is available on github.com/deepwlearning/deepwnetworks.

Table 1: Hyperparameters for the Mountain Car Environment.

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
γ	0.99	α	$1*10^{-3}$	β	0.4
ε^{Q}_{start}	0.95	ε^{Q}_{decay}	0.995	ε^{Q}_{min}	0.1
ε^{W}_{start}	0.99	ε^{W}_{decay}	0.9995	ε^{W}_{min}	0.1
ζ	0.6	τ^{Q}	$1*10^{-3}$	τ^{W}	$1*10^{-3}$
Batch size K	1024	Memory size M	$1*10^{4}$	Q Optimizer	Adam
W Optimizer	Adam	Q learning rate	$1*10^{-3}$	W learning rate	$1*10^{-3}$

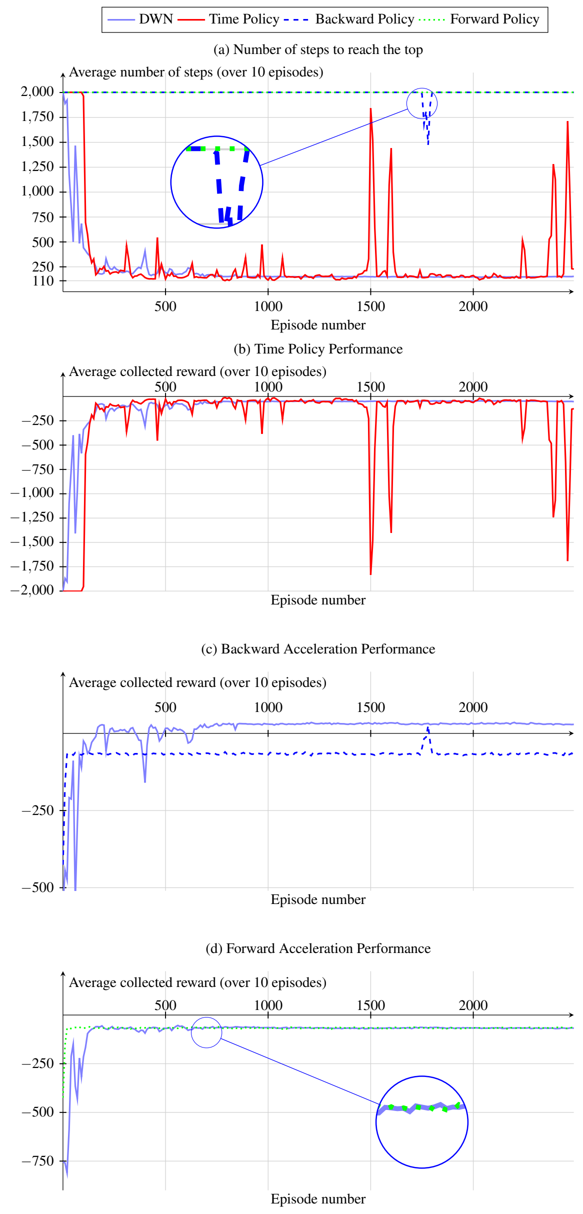

Figure 4: The comparison of performance between a policy as standalone DQN agent and a policy as a part of the proposed DWN.

by itself or part of DWN.

In Fig. 3 we show the percentage, i.e., how many times the DWN agent has selected an individual policy in an episode. At the start, the agent selects policies evenly. Such a behaviour is expected due to high starting epsilon-greedy value. However, as the epsilon decays with the number of episodes and Q-learning DQN learn, the agent starts to prefer one policy over the others. Interestingly, the backward accelerating policy proves to prevailing policy as the DWN selects it three times more often the other two policies combined.

In Fig. 4 we compare DWN with the performance of DQN when it receives only the reward of a particular policy. In Fig. 4 (a), we show the number of steps each policy requires to reach the top. Besides DWN and the DQN with time policy will reach the objective, i.e., arrive at the top of the hill. It appears, that when the reward signal is the only backward and forward policy the agent is unable, with a small exception, to learn to reach the top. However, when we look at the amount of collected reward a particular policy collects as part of DWN or individually is almost the same. Meaning, that without exception policies learn to maximise their rewards. Note that the difference of 100 in Fig. 4 (c) between DWN and backward acceleration policy is due to the reward signal. The agent receives a reward of 100 when it reaches the top, and the backward policy reaches the top only as part of DWN not individually.

Combining the gained insights from the above results, we demonstrate that DWN performs as expected: the agent is capable of reaching the end objective, while also maximise the reward collected by the individual policies within the DWN agent. In other words, policies in DWN are selected in such a way that on an individual level each policy can achieve its best performance, i.e., maximise its long-term rewards. An added benefit is that the agent can also reach the main objective, i.e., reach the top of the hill.

Table 2: Hyperparameters for the Deep Sea Environment.

Hyperparameter	Value	Hyperparameter	Value	Hyperparameter	Value
γ	0.9	α	$1*10^{-3}$	β	0.4
ε^{Q}_{start}	0.95	ε^{Q}_{decay}	0.995	ε^{Q}_{min}	0.25
ε^{W}_{start}	0.99	ε^{W}_{decay}	0.9995	ε^{W}_{min}	0.01
ζ	0.6	τ^{Q}	$1*10^{-3}$	τ^{W}	$1*10^{-3}$
Batch size K	1024	Memory size M	$1*10^{5}$	Q Optimizer	RMSprop
W Optimizer	RMSprop	Q learning rate	$1*10^{-3}$	W learning rate	$1*10^{-3}$

4.2 Deep Sea Treasure

The second environment, called *Deep Sea Treasure*, is a simple grid-world with treasure chests that increase in value the deeper they are. The deeper the chest is, the further away from the agent it is. The goal of this scenario is for the agent to learn to optimise for future rewards rather than opting for the fractional short-term gain. We used the environment as proposed and implemented in (Vamplew et al., 2011).

The deep sea environment has two objectives: time penalty and collected treasure. The time objective is for the agent to finish the episode, i.e., find the treasure, as quickly as possible. Therefore, the agent receives a negative reward of -1 at every step. The treasure reward depends on how deep is the treasure. Furthermore, the reward is increasing non-linearly with the depth and ranges from 1 to 124. In this scenario, our DWN agent has two policies: time and treasure. As in the previous environment, all ANN, i.e., $\theta^{Q}_{i}, \hat{\theta}^{Q}_{i}, \theta^{W}_{i}, \hat{\theta}^{W}_{i} \forall i$, have the same Convolutional Neural Network (CNN) structure. The first 2-dimensional convolution layer has three input channels and 16 output channels, and the second and third convolution layers have 32 channels. Every convolution layer has kernel size five with stride two, followed by batch normalisation. The last dense layer in the ANN has 1568 neurons. We list hyper-parameters in Table 2.

First, we analyse the performance of individual policies and compare it with DWN. For the individual policy, we employed DQN with the same neural structure as described above. In Fig. 5 we show the performance of three DQN solutions, each with different reward signal. The first DQN solution receives only the time penalty reward signal, the second only the treasure value signal, and the third the sum of the two reward signals. The DQN with only time reward learns to finish the episodes as fast as possible. Therefore, it learns to collect the first available treasure. The DQN with only treasure reward learns to collect the highest treasure reward of all approaches. However, it does not learn to collect the highest treasure rewards, i.e., 74 and 124. The performance of DQN with the sum of two rewards is exactly in the middle

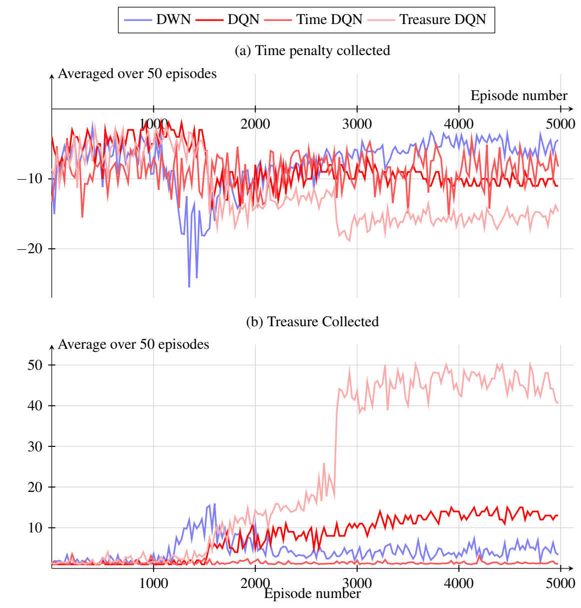

Figure 5: Time penalty and treasure collected, averaged over 50 episodes, for the number of episodes.

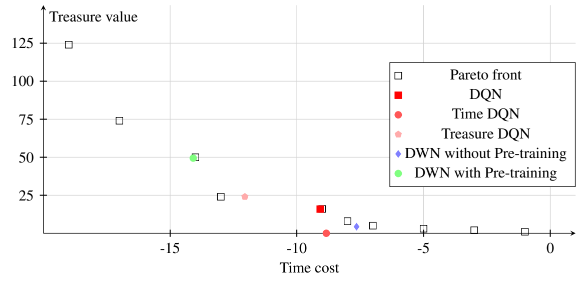

Figure 6: The Pareto front for the deep sea treasure environment.

of the two. Interestingly, the DWN performance is between the DQN with the sum reward and DQN with only time reward.

In Fig. 6 we show how close to the Pareto front the agent with a different approach can arrive after 5000 episodes. Note that results are the average of the last one hundred episodes. The DQN learns to reach the Pareto front. However, the collected treasure reward is far from ideal. The DWN, when trained from scratch, is close but under-performs in comparison to a DQN approach. Interestingly, DWN with pre-trained Q-networks finds a high treasure at the Pareto front. In the latter approach, we took advantage of DWN modularity properties. First, we trained time and treasure policy for 2500 episodes separately, and then for 2500 episodes, we trained as part of DWN. Such a behaviour can be explained that policies as part of DWN are not able to converge, thus giving them a head start, i.e., learning separately, can improve the DWN performance. Furthermore, such a result was expected because, as it was pointed out in the original W-learning paper, we need to allow the Q-learning networks to learn first.

5 Conclusion

In this paper, we have proposed a deep learning extension to W-learning, an approach that naturally resolves competition in multi-objective scenarios. We have demonstrated the proposed method's efficiency and superiority to a baseline solution in two environments: deep sea treasure and multi-objective mountain car. In both of these environments, the proposed DWN is capable of finding the Pareto front. Furthermore, we have also demonstrated the advantage of DWN modularity properties by showing that using a pre-trained policy can aid in finding the Pareto front in the deep sea treasure environment. In our future work, we will focus on improving the computational performance and evaluating the performance in more complex environments, e.g., SuperMarioBros (Kauten, 2018).

The proposed DWN algorithm can be employed in any system with multiple objectives such as traffic control, telecommunication networks, finance, etc. The condition being that each objective is represented with a different reward function. The main advantage of DWN is its ability to train multiple policies simultaneously. Furthermore, sharing the state space between policies is not mandatory, e.g., a policy for the mountain car environment policies could only need access to the velocity vector. Meaning that with DWN it is possible to train policies with different states due to the use of separate buffers for storing experiences.

ACKNOWLEDGEMENTS

This work was funded in part by the SFI-NSFC Partnership Programme Grant Number 17/NSFC/5224 and SFI under Frontiers for the Future project 21/FFP-A/8957.

REFERENCES

Abels, A., Roijers, D., Lenaerts, T., Nowé, A., and Steckelmacher, D. (2019). Dynamic weights in multi-objective deep reinforcement learning. In *International Conference on Machine Learning*, pages 11–20. PMLR.

Cardozo, N. and Dusparic, I. (2020). Learning run-time compositions of interacting adaptations. SEAMS '20, page 108–114, New York, NY, USA. Association for Computing Machinery.

Dusparic, I., Taylor, A., Marinescu, A., Cahill, V., and Clarke, S. (2015). Maximizing renewable energy use with decentralized residential demand response. In *2015 IEEE First International Smart Cities Conference (ISC2)*, pages 1–6.

Giupponi, L., Agusti, R., Pérez-Romero, J., and Sallent, O. (2005). A novel joint radio resource management approach with reinforcement learning mechanisms. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 621–626. Phoenix, AZ, USA.

Hribar, J., Marinescu, A., Chiumento, A., and DaSilva, L. A. (2022). Energy Aware Deep Reinforcement Learning Scheduling for Sensors Correlated in Time and Space. *IEEE Internet of Things Journal*, 9(9):6732–6744.

Humphrys, M. (1995). W-learning: Competition among selfish Q-learners.

Jin, Y. and Sendhoff, B. (2008). Pareto-Based Multiobjective Machine Learning: An Overview and Case Studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415.

Karlsson, J. (1997). *Learning to solve multiple goals*. University of Rochester.

Kauten, C. (2018). Super Mario Bros for OpenAI Gym. GitHub: github.com/Kautenja/gym-super-mario-bros.

Kusic, K., Ivanjko, E., Vrbanic, F., Greguric, M., and Dusparic, I. (2021). Spatial-temporal traffic flow control on motorways using distributed multi-agent reinforcement learning. *Mathematics - Special Issue Advances in Artificial Intelligence: Models, Optimization, and Machine Learning*, 9(23).

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Liu, C., Xu, X., and Hu, D. (2015). Multiobjective Reinforcement Learning: A Comprehensive Overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari With Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*.

Mossalam, H., Assael, Y. M., Roijers, D. M., and Whiteson, S. (2016). Multi-Objective Deep Reinforcement Learning. *arXiv preprint arXiv:1610.02707*.

Nguyen, T. T., Nguyen, N. D., Vamplew, P., Nahavandi, S., Dazeley, R., and Lim, C. P. (2020). A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence*, 96:103915.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *Presented at International Conference on Learning Representations (ICLR), San Diego, CA, May 7–9, 2015. arXiv preprint 1511.05952.*

Sprague, N. and Ballard, D. (2003). Multiple-goal reinforcement learning with modular sarsa(0). In *18th Int. Joint Conf. Artif. Intell.*, page 1445–1447.

Tajmajer, T. (2018). Modular multi-objective deep reinforcement learning with decision values. In *2018 Federated conference on computer science and information systems (FedCSIS)*, pages 85–93. IEEE.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84(1):51–80.

Wan, R., Zhang, X., and Song, R. (2020). Multi-objective reinforcement learning for infectious disease control with application to COVID-19 spread. *arXiv preprint arXiv:2009.04607*.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of Machine Learning Research (PMLR), vol.48*, pages 1995–2003. New York, USA.