Dynamic Weights in Multi-Objective Deep Reinforcement Learning

Axel Abels [1 2] **Diederik M. Roijers** [3] **Tom Lenaerts** [1 2] **Ann Nowé** [2] **Denis Steckelmacher** [2]

Abstract

Many real world decision problems are characterized by multiple conflicting objectives which must be balanced based on their relative importance. In the dynamic weights setting the relative importance changes over time and specialized algorithms that deal with such change, such as the tabular Reinforcement Learning (RL) algorithm by Natarajan & Tadepalli (2005), are required. However, this earlier work is not feasible for RL settings that necessitate the use of function approximators. We generalize across weight changes and high-dimensional inputs by proposing a multi-objective Q-network whose outputs are conditioned on the relative importance of objectives, and introduce Diverse Experience Replay (DER) to counter the inherent non-stationarity of the dynamic weights setting. We perform an extensive experimental evaluation and compare our methods to adapted algorithms from Deep Multi-Task/Multi-Objective RL and show that our proposed network in combination with DER dominates these adapted algorithms across weight change scenarios and problem domains.

1. Introduction

In *reinforcement learning (RL)* (Sutton & Barto, 1998), an agent learns to behave in an unknown environment based on the rewards it receives. In single objective RL these rewards are scalar. However, most real-life problems are more naturally expressed with multiple objectives. For example, autonomous drivers need to minimize travel time and fuel consumption, while maximizing safety (Xiong et al., 2016).

When user utility in a multi-objective problem is defined as a linear scalarization with weights per objective that are known in advance and fixed throughout learning and execution, the problem can be solved via single-objective RL. However, in many cases the weights can not be determined in advance (Roijers & Whiteson, 2017) or linear scalarization does not apply because the user utility cannot be expressed with a linear function (Moffaert & Nowé, 2014). In this paper, we focus on the setting where the weights are linear, but not fixed. Specifically, the parameters of the scalarization function change over time. For example, if fuel costs increase, a shorter travel time could no longer be worth the increased fuel consumption. This is called the *dynamic weights* setting (Natarajan & Tadepalli, 2005).

Many RL problems necessitate learning from raw input, e.g., images captured by cameras mounted on a car. Recently, Deep RL (Mnih et al., 2013) has enabled RL to be applied to problems where the input consists of images. However, most Deep RL research focuses on single-objective problems.

In this paper, we study the possibilities of Deep RL in the dynamic weights setting and show how transfer learning techniques can be leveraged to increase the learning speed by exploiting information from past policies. For tabular RL, these principles have previously been applied to, e.g., Buridan's ass problem (Natarajan & Tadepalli, 2005). However, because of its small and discrete state space this problem is not representative of complex real-world problems which often have vast or even continuous state spaces. In such complex problems, tabular RL is not feasible.

To tackle high-dimensional problems, we show that algorithms from related settings can be adapted to the dynamic weights settings but are inadequate. We therefore propose the *conditioned network (CN)*, in which a Q-Network is augmented to output weight-dependent multi-objective Q-value-vectors. To efficiently train this network, we propose an update rule specific to the dynamic weights setting. We further propose *Diverse Experience Replay (DER)*, to improve sample-efficiency and reduce replay buffer bias.

To benchmark the quality of our algorithms, we propose the first non-trivial high-dimensional multi-objective benchmark problem: *Minecart*. From raw visual input, an agent in Minecart must learn to adapt to the day's valuation of different resources to efficiently mine them while minimizing fuel consumption. We test the performance of our algorithms on

[1]Machine Learning Group, Université Libre de Bruxelles, Brussels, Belgium [2]Artificial Intelligence Lab, Vrije Universiteit Brussel, Brussels, Belgium [3]Computational Intelligence, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands. Correspondence to: Axel Abels <aabels@ulb.ac.be>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

two weight change scenarios and find that, while methods from related settings can be adapted to the dynamic weights setting, only our proposed CN can both quickly adapt to sparse abrupt weight changes and also converge to optimal policies when weight changes occur regularly. Furthermore, by maintaining a set of diverse trajectories, DER improves the performance of all tested algorithms.

2. Background

This section defines Markov Decision Processes and Q-Learning, then briefly reviews the Deep RL literature and Multi-objective RL.

2.1. Markov Decision Process

In RL, agents learn how to act in an environment in order to maximize their cumulative reward. A popular model for such problems is Markov Decision Processes (MDP), defined by a set of states S, a set of actions A, a transition function T which maps the state s_t and action a_t to a probability over all possible next states s_{t+1}, and a reward function R which maps each state $s \in S$ and action taken in it to an expected immediate reward $r_t = R(s_t, a_t)$. Under the standard assumption that future rewards are discounted by a factor $\gamma \in [0, 1]$, the goal of the agent is to find a policy $\pi^*(a|s)$ that maximizes the expected cumulative reward of the agent, i.e., its *return*, $g_T = \sum_{t=1}^{T} \gamma^{t-1} r_t$. We define a *trajectory* τ as a sequence of transitions from some state s_i to a state s_{j+1}; $\tau = [(s_i, a_i, r_i, s_{i+1}), ..., (s_j, a_j, r_j, s_{j+1})]$.

The value function $V^\pi : S \to \mathbb{R}$ of a policy π maps a state to the expected return obtained from that state, when π is followed, i.e., $V(s) = E_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1 = s]$. Correspondingly, the Q-function $Q^\pi : S \times A \to \mathbb{R}$ maps a state-action pair to the expected return obtained from that state when the action is executed, and then π is followed from the next state onwards. The value function V^* and Q-function Q^* that correspond to the optimal policy π^* are the *optimal* value functions. The optimal policy π^* can be computed from the optimal Q^* function; $\pi^*(a|s) = \mathbb{1}[a = \text{argmax}_{a'} Q^*(s, a')]$, i.e., the agent executes, at every time-step, the action whose Q-value in the current state is maximal. This is the *greedy policy* w.r.t. Q^*. The *stateless value* for a policy π is defined as $V^\pi = \sum_{s \in S} \mu(s) V^\pi(s)$, with $\mu(s)$ the probability distribution over initial states.

Q-Learning Q-learning (Watkins, 1989) is a reinforcement learning algorithm that allows an agent to learn Q^* for any (finite) MDP based on interactions with the environment. At every time-step, the agent observes the state s_t, executes a random action a_t with probability ε and $a_t \sim \pi(s_t)$ otherwise, receives a reward r_t, then observes the next state s_{t+1}. Based on this (s_t, a_t, r_t, s_{t+1}) *experience tuple*, the agent updates its estimate of Q^* at iteration k: $Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \delta_k$, where $\delta_k = r_t + \gamma \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a_t)$, with $\alpha > 0$ a small learning rate. Q-learning is proven to converge under reasonable assumptions (Tsitsiklis, 1994).

Deep Q-Learning (DQN) (Mnih et al., 2013) is a popular approach to generalize Q-Learning to high-dimensional environments. DQN approximates the Q-function by a neural network parameterized by θ. At every time step t, the (s_t, a_t, r_t, s_{t+1}) experience tuple is added to an experience buffer $\mathcal{D}$ and the Q-network is optimized on the loss $L_t(\theta_t)$ computed on a mini-batch of experiences:

$$L_t(\theta_t) = E_{(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{U}(\mathcal{D})} \left[(y_i(s_i, a_i) - Q(s_i, a_i; \theta_t))^2 \right]$$

with $y_i(s_i, a_i) = r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta_t^-)$ and θ^- the parameters of the *target network*. Training towards a fixed target network prevents approximation errors from propagating too quickly from state to state, and sampling experiences to train on (*experience replay*) increases sample efficiency and reduces correlation between training samples. *Prioritized experience replay* (Schaul et al., 2015b) improves training time by sampling transitions with large residual errors from which the agent can learn more.

2.2. Multi-objective RL

Multi-Objective MDPs (MOMDP) (White & Kim, 1980) are MDPs with a vector-valued reward function $\mathbf{r}_t = \mathbf{R}(s_t, a_t)$. Each component of $\mathbf{r}_t$ corresponds to one objective. A scalarization function f maps the multi-objective value $\mathbf{V}^\pi$ of a policy π to a scalar value, i.e., the user utility. In this paper we focus on linear f; each objective, i, is given a weight w_i, such that the scalarization function becomes $f(\mathbf{V}^\pi, \mathbf{w}) = \mathbf{w} \cdot \mathbf{V}^\pi$. An optimal solution for an MOMDP under linear f is a *convex coverage set (CCS)*, i.e., a set of undominated policies containing at least one optimal policy for any linear scalarization (Roijers et al., 2013). Depending on whether the focus is on asymptotic (Taylor & Stone, 2009) or cumulative performance, we distinguish Offline and Online Multi-Objective RL (MORL). In this paper we focus on Online MORL.

Offline MORL To learn vector-valued Q-functions for a given $\mathbf{w}$, *scalarized deep Q-learning (SDQL)* (Mossalam et al., 2016) extends the DQN algorithm to MORL, by modifying the loss:

$$L_t(\theta_t) = E_{(s_i, a_i, r_i, s_{i+1}) \sim \mathcal{U}(\mathcal{D})} \left[\frac{1}{N} \mathbb{1} \cdot (\mathbf{y}_i - \mathbf{Q}(s_i, a_i; \theta_t))^2 \right]$$

with $\mathbf{y}_i = \mathbf{r}_i + \gamma \mathbf{Q}(s_{i+1}, \text{argmax}_{a'}[\mathbf{Q}(s_{i+1}, a'; \theta_t^-) \cdot \mathbf{w}], \theta_t^-)$.

By sequentially training Q-networks until convergence on corner weights, they approximate the CCS with a set of Q-networks.

Online MORL Offline methods can be undesirable. In the *dynamic weights setting* for example, the weights of the scalarization function f can vary over time, and there is often not enough time to learn an entire CCS beforehand. Furthermore, the performance is evaluated with regards to the cumulative regret, i.e., the cumulative difference between the value the optimal policy would have obtained and the actual performance of the agent. In this setting, pre-training is not adequate, as it requires spending a lot of time training in anticipation rather than on the active weight vectors. Instead, the agent should learn, remember and apply policies on-the-fly as the weight vector changes. In tabular RL, Natarajan & Tadepalli (2005) have shown that instead of restarting training from scratch every time $\mathbf{w}$ changes, it is highly beneficial to continue learning from a previously learned policy. When the weight vector $\mathbf{w}$ changes to another value $\mathbf{w}'$, the policy π that was learned for $\mathbf{w}$ is stored in a set of policies Π, along with its value vector $\mathbf{V}^{\pi}$. As an initial policy for the new weight vector $\mathbf{w}'$, they select from Π the past policy with the highest scalarized value; $\pi_{init} = \text{argmax}_{\pi \in \Pi} \mathbf{V}^{\pi} \cdot \mathbf{w}'$.

Universal Value Function Approximators (UVFA) Schaul et al. 2015a build a single network capable of generalizing over multiple goals. Based on the observation that a goal is often a subset of the set of states, the network learns goal and state embeddings and uses a distance-based metric to combine both embeddings. This is achieved offline, by learning several value functions independently, factorizing embeddings and then training a network to approximate these values for any given goal. In MORL, a goal would be a specific weight vector and as such there is no clear relation between the goal (i.e., the importance of each objective) and the state. One could fall back on the concatenation of state and goal embeddings as suggested in (Schaul et al., 2015a), but they found this is prone to instability.

3. Contributions

Existing Deep (MO)RL algorithms are insufficient in the dynamic weights setting because they either build a complete set of policies in advance or spend a long time adapting to weight changes. We first propose our *Conditioned Network* method capable of generalizing multi-objective Q-values across weight vectors and then we propose *Diverse Experience Replay* to improve sample efficiency and counter the replay buffer's bias to recent weight vectors.

3.1. Conditioned Network (CN)

We propose our first main contribution, Conditioned Network (CN), in which a UVFA is adapted to output Q-value-vectors conditioned on an input weight vector (Figure 1). The training algorithm follows the standard DQN algorithm, i.e., the agent acts ε-greedily and stores its experiences in a replay buffer from which transitions are sampled to train the network on. Because the network also takes a weight-vector

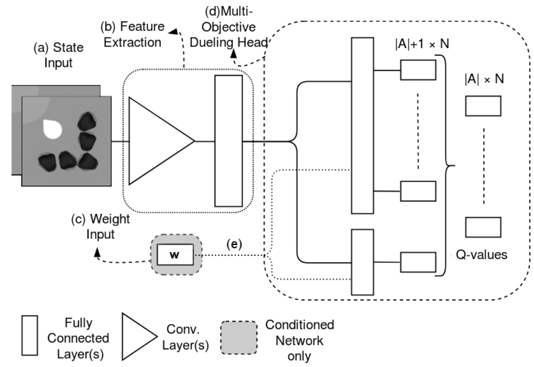

Figure 1: Features are extracted from the raw input by convolutional layers followed by a fully connected layer. The extracted features (output of (b)) are fed into an N objectives Dueling DQN head (d). The conditioned architecture feeds a weight input (c) into the Q-value head (link (e)).

as input, the selection of weight vectors to train the network on requires additional consideration.

While generalization is important, attention should be paid to the active weight vector, such that the agent can quickly perform well for the objectives that are important at the moment. However, if we do not maintain trained policies, the network may overfit to the current region of the weight space and forget past policies. To avoid this overfitting, we propose that samples should be trained on more than one weight vector at a time. Specifically, to promote quick convergence on the new weight vector's policy and to maintain previously learned policies, each experience tuple in a mini-batch is updated w.r.t. the current weight vector and a random previously encountered weight vector. Given a mini-batch of B transitions, we compute the loss for a given transition $(s_j, a_j, \mathbf{r}_j, s_{j+1})$ as the sum of the loss on the active weight vector $\mathbf{w}_t$ and on $\mathbf{w}_j$ randomly sampled from the set of encountered weights.

$$\frac{1}{2}\left[|\mathbf{y}^{(j)}_{\mathbf{w}_t} - \mathbf{Q}_{CN}(a_j, s_j; \mathbf{w}_t)| + |\mathbf{y}^{(j)}_{\mathbf{w}_j} - \mathbf{Q}_{CN}(a_j, s_j; \mathbf{w}_j)|\right]$$

$$\mathbf{y}^{(j)}_{\mathbf{w}} = \mathbf{r}_j + \gamma \mathbf{Q}^{-}_{CN}(\underset{a \in A}{\text{argmax}}\, \mathbf{Q}_{CN}(a, s_{j+1}; \mathbf{w}) \cdot \mathbf{w}, s_{j+1}; \mathbf{w})$$

where $\mathbf{Q}_{CN}(a, s; \mathbf{w})$ is the network's Q-value-vector for action a in state s and weight vector $\mathbf{w}$. Training the same sample on two different weight vectors has the added advantage of forcing the network to identify that different weight vectors can have different Q-values for the same state. Please see Appendix 1.3 for a detailed description of the CN algorithm.

This method deviates from UVFA on three major points. First, the outputs of our network are multi-objective, secondly, the whole network is trained end-to-end, and finally, we stabilize learning through our update rule which is adapted to the dynamic weights setting.

3.2. Diverse Experience Replay

A particular challenge to using experience replay for Deep MORL is that an experience buffer obtained through a weight vector's optimal policy can be harmful to another weight vector's training process. Existing offline approaches circumvent this by resetting the replay buffer when the trained policy changes and restarting the exploration phase. However, excessive exploration harms cumulative performance in the (online) dynamic weights setting. To ensure the agent learns adequately, the replay buffer must contain experiences relevant[1] to any future weight vector's optimal policy. This not the case when using a standard replay buffer, as it is biased towards recently encountered weight vectors. A policy $\pi^{\mathbf{w}}$ trained exclusively on experiences obtained through another policy $\pi^{\mathbf{w}'}$ will typically diverge from the optimal policy for $\mathbf{w}$. Making the replay buffer larger such that early experiences obtained through random exploration are still present is impractical for two reasons; (1) unless the replay buffer is infinite, older experiences could still be erased before reaching areas of the weight-space which need them. And (2), even if these relevant experiences are still present, they could be vastly outnumbered. Therefore, we propose a different solution to consistently provide relevant experiences to a learner for any weight vector.

We propose *Diverse Experience Replay (DER)*, a diverse buffer from which relevant experiences can be sampled for weight vectors whose policies have not been executed recently. DER replaces standard recency-based replay by diversity-based memorization. Furthermore, instead of considering each transition independently, DER handles trajectories as atomic units. To understand why, consider a trajectory of experiences from initial state to terminal state. The absence of an experience between initial and terminal state can make the task of propagating Q-values from the terminal to the initial state infeasible, as the learner has to infer the missing link. When using standard replay buffers this is not an issue, as experiences are added and removed sequentially. Hence, for the vast majority of experiences in standard replay buffers, the preceding and subsequent transitions are also present. To avoid partial trajectories, we treat trajectories as atomic units when considering them for addition in, or deletion from the diverse buffer. To reduce the computational cost of comparing trajectories, we compute a signature for each trajectory on which we enforce diversity. This signature can for example be the trajectory's discounted cumulative reward (i.e., its return), or the set of perceptual hashes (Zauner, 2010) of the trajectory's frames. Specifically, when a new trajectory is considered for addition into a diverse buffer $\mathcal{D}'$, each trajectory's signature is computed by a signature function $\mathbf{s}$. A diversity function

[1]Relevant experiences are experiences that can be expected to help a learner converge towards the optimal policy.

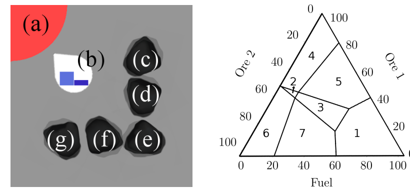

Figure 2: **Left:** Instance of the Minecart environment with 5 mines ((c) to (g)) containing varying amounts of 2 ores. The 2 bars on the minecart (b) indicate how much of each ore is present in the cart. Ores are sold on the base (a). **Right:** Weight vectors in the same region share the same optimal policy. Axes are the relative importance in % of each objective. We distinguish (1) collecting no resources if the fuel cost is too high, (6,7) privileging ore 2, (4,5) privileging ore 1, and (2,3) privileging the quick collection of either ore. Differences between each pair lie in a higher fuel cost, in which case it is optimal to accelerate less.

d then computes the relative diversity of each trajectory's signature. The new trajectory is only added to the diverse buffer if its inclusion increases the overall diversity of the diverse buffer. When it is full, the traces that contribute least to diversity are ejected from the diverse buffer.

Diversity in the Dynamic Weights Setting In this setting we wish to maintain a set of trajectories relevant to any region of the weight-space by ensuring trajectories with a wide variety of future rewards are present. To achieve this, we propose to; (1) treat an episode's transitions as one trajectory, (2) use a trajectory's return vector as its signature $\mathbf{s}(\tau) = \sum_{t=0}^{|\tau|} \gamma^t \mathbf{r}_t$ and (3) use a metric from multi-objective evolutionary algorithms, called the *crowding distance* (Deb et al., 2002), as a diversity function. Applied to return vectors, the crowding distance promotes the presence of trajectories spread across the space of returns.

We also maintain a standard FIFO buffer to which experiences are added first. When this buffer is full, the oldest trajectory in it is removed and considered for addition in the diverse buffer. These two buffer types allow a new weight vector's policy to be bootstrapped on experiences from the diverse buffer and then further trained on the experiences it progressively adds to the standard buffer. Please see Appendix 1.5 for a detailed description of the DER algorithm.

3.3. Minecart Problem

Existing Deep MORL problems, such as the image version of Deep Sea Treasure (DST) (Mossalam et al., 2016), are relatively trivial, i.e., it has 4 actions and even though the states are presented as an image, the number of actually distinct states is only around $\sim$50. This is in stark contrast with single-objective Deep RL for which among others, ALE (Bellemare et al., 2012) provides a diverse set of challenging environments. To close this gap, we propose an original benchmark, the Minecart problem[2]. Minecart has a continuous state space, stochastic transitions and delayed rewards. The Minecart environment consists of a rectangular image, depicting a base, mines and the minecart controlled by the agent. A typical frame of the Minecart environment is given in Figure 2 Left. Each episode starts with the agent on top of the base. Through the *accelerate*, *brake*, *turn left*, *turn right*, *mine*, or *do nothing* actions, the agent should reach a mine, collect resources and return to the base to sell them.

The reward vectors are N-dimensional: $\mathbf{r} = (r_1, ..., r_N)$. The first $N-1$ elements correspond to the amount of each of the $N-1$ resources the agent sold, the last element is the consumed fuel. Particular challenges of this environment are the sparsity of the first $N-1$ components of the reward vector, as well as the delay between actions (e.g., mining) and resulting rewards. The resources an agent collects by mining are generated from the mine's random distribution, resulting in a stochastic transition function. All other actions are deterministic. The weight vector $\mathbf{w}$ expresses the relative importance of the objectives, i.e., the price per resource. For the default configuration of the Minecart, the weight-space has 7 regions with a different optimal policy (Figure 2, right). A full description of the environment and default parameter values is given in the appendix. In the dynamic weights Minecart problem, an agent should quickly adapt to fluctuations in the price of resources.

4. Adapted Algorithms

For completeness, we show how methods from related settings can be adapted to dynamic weights, but are suboptimal.

4.1. UVFA

We first present how we adapted UVFA. To avoid expensive pre-training, we consider as basis the direct bootstrapping variant of UVFA and train the network end-to-end. Because a distance based metric is not applicable in our setting[3], we concatenate the state features and the goal (i.e., weight-vector) and feed them into the policy heads. The network thus shares the same overall architecture as CN but outputs scalar Q-values. Following UVFA's direct bootstrapping, we sample goals and transitions from the replay buffer to train the network on. For each transition $(s_j, a_j, \mathbf{r}_j, s_{j+1})$ of a mini-batch, we sample $\mathbf{w}_j$ from the set of encountered weights and minimize the loss $|y_j - Q(a_j, s_j; \mathbf{w}_j)|$, with

$$y_j = \mathbf{r}_j \cdot \mathbf{w}_j + \gamma Q^{-}(\underset{a \in A}{\operatorname{argmax}}\, Q(a, s_{j+1}; \mathbf{w}_j), s_{j+1}; \mathbf{w}_j)$$

[2]The code can be found at https://github.com/axelabels/DynMORL

[3]In our case, the goal (a specific $\mathbf{w}$) is not comparable to a subset of states.

$Q(a, s; \mathbf{w})$ is the network's Q-value for action a in state s and weight vector $\mathbf{w}$. Setting $g = \mathbf{w_j}$ and replacing $\mathbf{r}_j \cdot \mathbf{w}_j$ by $R_g(s_j, a_j, s_{j+1})$ in the above equations gives an equivalent goal-oriented notation as in (Schaul et al., 2015a).

4.2. Multi-Network (MN)

Combining existing work on tabular dynamic weights (Natarajan & Tadepalli, 2005) and multi-objective deep RL for different settings (Mossalam et al., 2016), we propose to gradually build a set of policies represented by MO Q-networks, Π. Key insights of this approach are that; (1) for a given $\mathbf{w}$ we can train a Q-network for a region of the weight-space around $\mathbf{w}$, (2) by training multiple Q-networks on different weight vectors we can cover more regions of the weight-space, and (3) we can speed up learning by knowledge transfer from previously trained neural networks.

By only storing un-dominated Q-networks (i.e., Q-networks that are optimal for at least one encountered weight vector[4]), we gradually approximate Π, a subset of the CCS relevant to the encountered weights. Because a CCS is typically relatively small, the number of networks we need to train and maintain in memory is also expected to be small.

Each policy $\pi_\mathbf{w}$ is trained for the active weight vector $\mathbf{w}$ following *scalarized deep Q-learning* (Mossalam et al., 2016). When the active weights change, the stateless value of the policy $\pi_\mathbf{w}$, $\mathbf{V}^{\pi_\mathbf{w}}$, is compared to all previously saved policies. If $\mathbf{V}^{\pi_\mathbf{w}}$ improves upon the maximum scalarized value of the policies already in Π for at least one past weight vector or for the current weight vector $\mathbf{w}$, it is saved, otherwise it is discarded. To limit memory usage and ensure fast retrieval by keeping Π small, all old policies made redundant by $\pi_\mathbf{w}$ are removed from Π. A policy is redundant if it is not the best policy for any encountered weight vector.

We hot-start learning for each new $\mathbf{w}$ by copying the policy $\pi' \in \Pi$ whose scalarized value $\mathbf{V}^{\pi'} \cdot \mathbf{w}$ is maximal. Following previous transfer learning approaches (Mossalam et al., 2016; Parisotto et al., 2015; Du et al., 2016), MN copies parameters from a source network (π''s Q-network) to the current policy's Q-network. Because MN compares policies based on predicted Q-values, inaccurate outputs disturb training by biasing MN to overestimated policies. As a result, MN needs long training times for each weight vector to obtain accurate values to compare. Please see Appendix 1.4 for a detailed description of the MN algorithm.

5. Experimental Evaluation

We test the performance of our algorithms on two different problems: the image version of Deep Sea Treasure (DST)

[4]By encountered weight vectors we mean the set of weight vectors the agent has experienced since it started learning.

proposed by Mossalam et al. (2016), and our newly proposed benchmark, the Minecart problem. Moreover, we use two weight change scenarios. We first evaluate the performance when weight changes are sparse, as in (Natarajan & Tadepalli, 2005), in which case an agent (and its replay buffer) could overfit to the active weights. Then, we look at regular weight changes, in which case it can be tempting to learn a policy that is good for most weights but optimal for none. We compare CN against MN, UVFA, a Multi-Objective DQN trained on the current $\mathbf{w}$ only (MO), and two ablated versions of CN, CN-ACTIVE and CN-UVFA.

5.1. Experimental Setup

First, we evaluate the performance for *sparse* and large weight changes; the current weight, $\mathbf{w}$, is randomly sampled from a Dirichlet distribution ($\alpha = 1$) every $50k$ steps for Minecart and $5k$ steps for DST. Second, we test on *regular* weight changes; $\mathbf{w}$ linearly moves to a random target, $\mathbf{w}'$, over 10 episodes, after which a new $\mathbf{w}'$ is sampled. Both variants are evaluated on the Minecart environment, and on an image version of Deep Sea Treasure (DST, fully described in the appendix).

We evaluate policies based on their *regret*, i.e., the difference between optimal value and actual return, $\Delta(\mathbf{g}, \mathbf{w}) = \mathbf{V}^*_{\mathbf{w}} \cdot \mathbf{w} - \mathbf{g} \cdot \mathbf{w} = \mathbf{V}^*_{\mathbf{w}} \cdot \mathbf{w} - \sum_{t=0}^{T} \gamma^t \mathbf{r_t} \cdot \mathbf{w}$, where $\mathbf{g}$ is the discounted cumulative reward, $\mathbf{V}^*_{\mathbf{w}}$ denotes the optimal value for $\mathbf{w}$, $\{\mathbf{r}_0, ..., \mathbf{r}_T\}$ is the set of vector-valued rewards collected during an episode of length T. Unlike the return, the regret allows for a common optimal value regardless of the weights, i.e., an optimal policy always has 0 regret. This is a necessary condition to consistently evaluate performance over different runs and for different weight vectors.

We include the performance of the adapted algorithms we proposed, the MN algorithm as well as UVFA. To show the benefits of our proposed loss for CN we perform an ablation study by also (1) training only on the active weight vector (CN-ACTIVE) and (2) training only on randomly sampled weight vectors (CN with UVFA loss, CN-UVFA). As a baseline, we use a basic *Multi-Objective DQN approach (MO)*; a single multi-objective DQN continuously trained on only the current $\mathbf{w}$ through scalarized Deep Q-learning. MO does not maintain multiple networks and the weight vector is not fed as input to the network. An alternative naive baseline for general MORL purposes suggested by (Liu et al., 2015) learns optimal Q-values for each objective then selects actions by scalarizing these multiple single-objective Q-values. Because the resulting Q-value-vectors do not capture the necessary trade-offs, this baseline can only perform in edge cases where one objective outweighs all others. As a result it performed poorly in our tests and we restrict its experimental results to the appendix.

All algorithms are run with and without DER and with prioritized sampling (Schaul et al., 2015b).

5.2. Results

Results for each weight change scenario are collected over 10 runs. Plots are smoothed by averaging over 200 steps.

5.2.1. Sparse weight changes

We determine how robust our algorithms are to overfitting to recent weight vectors by evaluating the performance for few but large weight changes (Left plots, Figures 3 and 4). Here, the main challenges are that (1) the agent's policy could overfit to the current $\mathbf{w}$ and forget policies for past weight vectors, and (2) the replay buffer could be biased towards experiences for recent $\mathbf{w}$'s.

Minecart As the MO baseline is unable to remember previously learned policies, it must repeatedly (re-)learn policies, leading to a loss in performance whenever weight changes occur. Moreover, the replay buffer bias prevents the MO agent from efficiently converging to new optimal policies. Using DER helps in this respect. The middle plot in Figure 3 illustrates the effect of having a secondary diverse buffer (DER). While recent experiences (orange) are concentrated in the same region, the diverse experiences (blue) are spread across the space of possible returns. By storing trained policies, MN can continue learning from the best policy in memory for each new $\mathbf{w}$. However, if no relevant experiences are in the buffer, it can be unable to optimize for the new weights. Thus, as for MO, the inclusion of DER significantly improves performance. While MN learns more slowly than other algorithms it is on par with the best performing algorithms over the last 250k steps when using DER, as it takes time to train a suitable set of policies.

Comparing CN to MN, we find that their performance is similar without DER. In addition to the difficulty of learning a new policy without diversity, CN is also susceptible to forgetting learned policies if the replay buffer is biased towards another policy. DER solves both problems and significantly improve performance. We find that over the last 250k steps, CN's performance with DER is not significantly different from MN's performance with DER. However, overall performance does improve over MN. We thus conclude that while MN and CN both ultimately learn a good set of policies, CN does so quicker. Additionally, we find that CN with our proposed loss outperforms the alternatives. Specifically, training uniformly on weight vectors sampled from the set of encountered weight vectors (CN-UVFA) significantly hurts performances without DER. By not consistently training on the current $\mathbf{w}$, CN-UVFA puts more effort into maintaining old policies than into learning new policies. As a result, it takes longer for the agent to perform well for new weight vectors, and relevant experiences are less likely to be collected. Hence, slower convergence leads to fewer

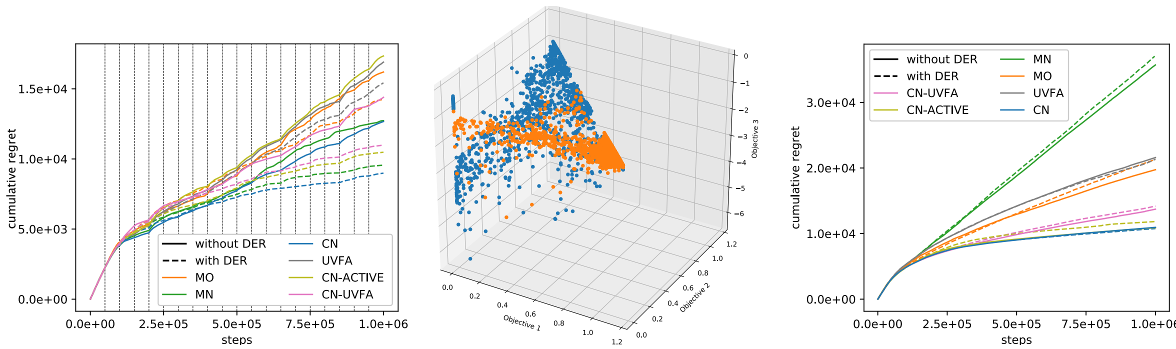

Figure 3: Solid lines plot performance without DER, dashed lines plot the performance with DER. **Left:** Cumulative regret for the Minecart problem when weights change every 50k steps (vertical lines), MN+DER and CN-UVFA+DER overlap eachother. **Middle:** Effect of DER on the replay buffer's content, each dot represents a trajectory's return vector. The non-diverse buffer (orange dots) is biased towards the recent weight-vector (favoring objective 1). The diverse buffer (blue dots) maintains a set of returns spread across the space of possible returns. **Right:** Cumulative regret for the Minecart problem when weights change over the span of 10 episodes, CN, CN+DER and CN-ACTIVE overlap in the lowest curve.

Table 1: Average episodic regret (Mean Δ) and improvement over MO with Standard ER baseline (>baseline) for both weight change scenarios (lower is better). We distinguish overall performance and performance over the last 250k steps.

		Overall				Last 250k steps			
		Standard ER		DER		Standard ER		DER	
	Algorithm	Mean Δ	>baseline	Mean Δ	>baseline	Mean Δ	>baseline	Mean Δ	>baseline
Sparse Weight Changes	MO	0.324	–	0.285	-12.04%	0.275	–	0.207	-24.73%
	MN	0.255	-21.3%	0.191	-41.05%	0.139	-49.45%	**0.063**	**-77.09%**
	CN	0.253	-21.91%	**0.18**	**-44.44%**	0.184	-33.09%	0.068	-75.27%
	CN-UVFA	0.288	-11.11%	0.22	-32.1%	0.218	-20.73%	0.102	-62.91%
	CN-ACTIVE	0.347	+7.1%	0.21	-35.19%	0.316	+14.91%	0.088	-68.0%
	UVFA	0.338	+4.32%	0.308	-4.94%	0.302	+9.82%	0.253	-8.0%
Regular Weight Changes	MO	0.398	–	0.43	+8.04%	0.258	–	0.319	+23.64%
	MN	0.718	+80.4%	0.746	+87.44%	0.67	+159.69%	0.709	+174.81%
	CN	0.222	-44.22%	**0.219**	**-44.97%**	0.069	-73.26%	**0.064**	**-75.19%**
	CN-UVFA	0.278	-30.15%	0.287	-27.89%	0.149	-42.25%	0.149	-42.25%
	CN-ACTIVE	0.221	-44.47%	0.24	-39.7%	0.065	-74.81%	0.071	-72.48%
	UVFA	0.435	+9.3%	0.43	+8.04%	0.273	+5.81%	0.267	+3.49%

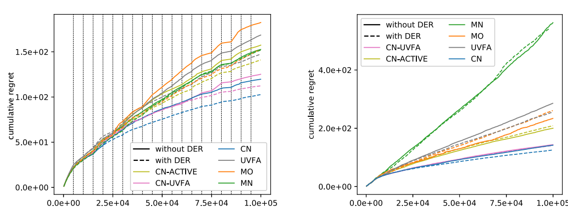

Figure 4: Cumulative regret for DST. Solid lines represent performance without DER, dashed lines with DER. **Left:** sparse weights changes, every 5K steps (vertical lines). **Right:** regular weight changes over the span of 10 episodes. CN and CN-UVFA(+DER) overlap near the bottom.

relevant experiences, in turn leading to slower convergence. When we include DER, relevant experiences are present despite the slower convergence, leading to a smaller impact on performance. UVFA shares the same flawed weight selection as CN-UVFA, and in addition only outputs scalar Q-values, meaning it does not exploit the added structure provided by the multi-objective rewards. These two factors in combination with the single-goal loss lead to performance close to our MO baseline. When we only train the Conditioned Network on the active weight vector (CN-ACTIVE), there is no explicit mechanism to preserve past policies, as a result CN-ACTIVE is likely to overfit to the current $\mathbf{w}$. CN-ACTIVE is outperformed in overall and final performance by MN, CN as well as by MO and CN-UVFA without DER.

DST We find that, while CN+DER still performs best for the DST problem, the performance of other algorithms is permuted. While the relative performances of MN, CN and CN-UVFA seem similar to those we obtained for the minecart problem, we found that CN-ACTIVE and MN perform relatively worse. What's more, DER seems to have no significant impact on the performance of MN. We hypothesize that MN performs worse for DST than for Minecart because the smaller distance between optimal policies in DST is harder to distinguish from approximation errors.

5.2.2. Regular weight changes

When weights change quickly, agents could fail to converge in time, resulting in sub-optimal policies for most weights.

Minecart As the rightmost plot in Figure 3 illustrates, regular weight changes lead to significantly different results w.r.t. sparse weight changes. While there is a slight loss in performance when adding DER to MO, there is no qualitative difference, as we found that with or without DER, MO converges to a single policy and applies it for all weight vectors. In contrast, CN learns to perform close to optimally for all weight vectors. Because CN continuously trains a single network towards multiple policies, its training process is not affected by the regular changes. In Minecart, when weights change regularly, CN-ACTIVE does not have enough time to overfit, resulting in performance on par with CN. CN-UVFA's performance remains poor, suggesting that emphasizing training on the current weight vector is crucial in Minecart. UVFA's performance is again close to MO, confirming it is not suited to the online dynamic weights setting. Due to the short per-weight training times, the networks in MN do not have enough time to converge for any given weight vector. As a result, their outputs, on which selection is done, are inaccurate. This makes MN discard more accurate newer policies in favor of older overestimated policies, and ultimately prevents it from learning. In contrast to sparse weight changes, there is no significant benefit to using DER as, due to the regular small weight changes, relevant experience is still in the replay buffer for new $\mathbf{w}$.

DST We obtained similar results for CN in DST. However, CN-ACTIVE performs worse for DST (-15%) than for Minecart (-44%). We hypothesize that when the distance between optimal policies is large (as in Minecart) focus should be put on the active weight vector to close the gap to the new optimal policy. Conversely, when optimal policies are close together (as in DST), unmaintained policies can more easily diverge from an optimal policy to a near-optimal policy.

In summary, our new algorithm CN dominates all other algorithms (with and without DER). We conclude that our proposed loss balances between learning new policies and maintaining learned policies well. Furthermore, MN is only able to perform well when given enough training time to learn accurate Q-values. Finally, DER improves performance when diversity cannot be expected to occur naturally.

6. Related Work

Natarajan & Tadepalli (2005) introduce the dynamic weights setting and show how it can be solved for low-dimensional problems by training a set of policies through tabular RL. The MN algorithm shares the same ideas, but addresses the additional challenges of Deep RL. Similar to MN, DOL (Mossalam et al., 2016) solves an image version of DST for the (off-line MORL) unknown weights scenario rather than the (on-line MORL) dynamic weights scenario. DOL builds a CCS in which each policy is implemented by a DQN. However, (1) the solved problem has a small underlying state-space while our Minecart problem is continuous, (2) weights chosen by DOL can be trained upon as long as necessary, and (3) only the final performance matters. Selective Replay (Isele & Cosgun, 2018) prevents catastrophic forgetting in single-objective multi-task problems. Similarly, (De Bruin et al., 2018) propose alternatives to FIFO buffers for single-objective Deep RL. Neither solution factors in the challenges we addressed with DER (e.g., long-term dependencies between experiences which we handle by storing trajectories) and hurt performance in this setting (please see the appendix for an experimental comparison). Successor Features (SF) (Barreto et al., 2016) decomposes a scalar reward into a product of state features and task weights to enable transfer learning between tasks. While these two components are analogous to the multi-objective reward and weight vectors, our work focuses on learning when this decomposition is given rather than learning the decomposition. Removing this decomposition learning from the proposed *SFQL* reduces it to an algorithm similar to *MN*. Universal Successor Features Approximators (Borsa et al., 2019) and Universal Successor Representations (Ma et al., 2018) combine the benefits of SF and UVFA to further generalize across goals. As for SF and UVFA separately, the challenges of Online MORL are not addressed.

7. Conclusion and Future Work

In this paper, we proposed the CN algorithm capable of tackling high-dimensional dynamic weights problems by learning weight-dependent multi-objective Q-values. We identified the drawbacks of FIFO experience replay for dynamic weights and proposed DER, which maintains a set of trajectories such that any policy can benefit from experiences present in this secondary buffer. To evaluate the performance of our algorithms we introduced the high-dimensional, continuous and stochastic Minecart problem. Our results show that CN dominates adapted algorithms from related settings in different weight change scenarios. Furthermore, our proposed loss, on the active weight vector and a random past weight vector, enables the network to generalize across weight vectors. On Minecart and DST we showed that CN always comes close to optimality, while MN fails to converge when weights regularly change.

In future work, we aim to integrate additional transfer learning techniques to further promote knowledge re-use between weight vectors. Finally, we aim to explore DER variants.

References

Barreto, A., Munos, R., Schaul, T., and Silver, D. Successor features for transfer in reinforcement learning. *CoRR*, abs/1606.05312, 2016. URL http://arxiv.org/abs/1606.05312.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D. J., van Hasselt, H., Munos, R., Silver, D., and Schaul, T. Universal successor features approximators. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1VWjiRcKX.

De Bruin, T., Kober, J., Tuyls, K., and Babuška, R. Experience selection in deep reinforcement learning for control. *The Journal of Machine Learning Research*, 19(1):347–402, 2018.

Deb, K., Pratap, A., Agarwal, S., and A. M. T. Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. 6:182 – 197, 05 2002.

Du, Y., de la Cruz, Jr., G. V., Irwin, J., and Taylor, M. E. Initial Progress in Transfer for Deep Reinforcement Learning Algorithms. In *Proceedings of Deep Reinforcement Learning: Frontiers and Challenges workshop (at IJCAI)*, New York City, NY, USA, July 2016.

Isele, D. and Cosgun, A. Selective experience replay for lifelong learning. *CoRR*, abs/1802.10269, 2018. URL http://arxiv.org/abs/1802.10269.

Liu, C., Xu, X., and Hu, D. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, March 2015. ISSN 2168-2216. doi: 10.1109/TSMC.2014.2358639.

Ma, C., Wen, J., and Bengio, Y. Universal successor representations for transfer reinforcement learning. *CoRR*, abs/1804.03758, 2018. URL http://arxiv.org/abs/1804.03758.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Moffaert, K. V. and Nowé, A. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15:3663–3692, 2014. URL http://jmlr.org/papers/v15/vanmoffaert14a.html.

Mossalam, H., Assael, Y. M., Roijers, D. M., and Whiteson, S. Multi-objective deep reinforcement learning. *CoRR*, abs/1610.02707, 2016. URL http://arxiv.org/abs/1610.02707.

Natarajan, S. and Tadepalli, P. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pp. 601–608, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102427. URL http://doi.acm.org/10.1145/1102351.1102427.

Parisotto, E., Ba, L. J., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015. URL http://arxiv.org/abs/1511.06342.

Roijers, D. M. and Whiteson, S. Multi-objective decision making. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 11(1):1–129, 2017.

Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. A survey of multi-objective sequential decision-making. *J. Artif. Int. Res.*, 48(1):67–113, October 2013. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=2591248.2591251.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015a. PMLR. URL http://proceedings.mlr.press/v37/schaul15.html.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015b. URL http://arxiv.org/abs/1511.05952.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford book. Bradford Book, 1998. ISBN 9780262193986. URL https://books.google.be/books?id=CAFR6IBF4xYC.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

Tsitsiklis, J. N. Asynchronous stochastic approximation and q-learning. *Machine Learning*, 16(3):185–202, Sep 1994. ISSN 1573-0565. doi: 10.1023/A:1022689125041. URL https://doi.org/10.1023/A:1022689125041.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. Empirical evaluation methods for multi-objective reinforcement learning algorithms. *Machine Learning*, 84(1):51–80, Jul 2011. ISSN 1573-0565. doi: 10.1007/s10994-010-5232-5. URL https://doi.org/10.1007/s10994-010-5232-5.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *AAAI*, pp. 2094–2100, 2016.

Wang, Z., de Freitas, N., and Lanctot, M. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL http://arxiv.org/abs/1511.06581.

Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989. URL http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf.

White, C. C. and Kim, K. M. Solution procedures for solving vector criterion Markov decision processes. *Large Scale Systems*, 1:129–140, 1980.

Xiong, X., Wang, J., Zhang, F., and Li, K. Combining deep reinforcement learning and safety based control for autonomous driving. *CoRR*, abs/1612.00147, 2016. URL http://arxiv.org/abs/1612.00147.

Zauner, C. Implementation and benchmarking of perceptual image hash functions. 2010.