Inherently Interpretable Deep Reinforcement Learning Through Online Mimicking

Andreas Kontogiannis[1,2](✉) and George A. Vouros[1]

[1] University of Piraeus, Piraeus, Greece
andr.kontog@gmail.com
[2] National Technical University of Athens, Athens, Greece

Abstract. Although deep reinforcement learning (DRL) methods have been successfully applied in challenging tasks, their application in real-world operational settings - where transparency and accountability play important roles in automation - is challenged by methods' limited ability to provide explanations. Among the paradigms for explainability in DRL is the interpretable box design paradigm, where interpretable models substitute inner closed constituent models of the DRL method, thus making the DRL method "inherently" interpretable. In this paper we propose a generic paradigm where interpretable DRL models are trained following an online mimicking paradigm. We exemplify this paradigm through XDQN, an explainable variation of DQN that uses an interpretable model trained online with the deep Q-values model. XDQN is challenged in a complex, real-world operational multi-agent problem pertaining to the demand-capacity balancing problem of air traffic management (ATM), where human operators need to master complexity and understand the factors driving decision making. XDQN is shown to achieve high performance, similar to that of its non-interpretable DQN counterpart, while its abilities to provide global models' interpretations and interpretations of local decisions are demonstrated.

Keywords: Deep Reinforcement Learning · Mimic Learning · Explainability

1 Introduction

Deep Reinforcement Learning (DRL) has mastered decision making policies in various difficult control tasks [11], games [19] and real-time applications [31]. Despite the remarkable performance of DRL models, the knowledge of mastering these tasks remains implicit in deep neural networks (DNNs). Thus, its application in real-world operational settings is challenged by methods' limited ability to provide explanations at global (policy) and local (individual decisions) levels. This lack of interpretability makes it difficult for human operators to understand DRL solutions, which can be important for solving safety-critical real-world tasks. Additionally, DRL models are unable to provide information about the evolution of models during the training process, which is useful to gain insights about the models' accumulated knowledge through time.

To address the aforementioned challenges, one may follow different paradigms for the provision of explanations: the interpretable box design paradigm is one of them where interpretable models substitute or are integrated to inner closed-box components of DRL [29]. Mimic learning has been proposed so as to infer interpretable models that mimic the behavior of well-trained DNNs [1,4]. In the DRL case, to improve interpretability, mimicking models can replace closed-box DRL models [16,29]. To do so, mimicking models must achieve performance that is comparable to closed-box models, also optimizing their *fidelity* [29], so that interpretable and closed models take the same decisions for the same reasons, in any specific circumstance. To extract knowledge from close-box models, recent works (e.g. [2,16]) have applied mimic learning using decision trees: In this case, criteria used for splitting tree nodes provide a tractable way to explain the predictions made by the controller.

Typically, mimic learning approaches require well-trained networks (which we refer to as *mature* networks), whose behaviour are mimicking towards interpretability. In doing so, interpretability of the model during the training process is totally ignored. In real-world settings this could be quite impractical, since the training overhead required to train the mimic models can often be a sample-inefficient and time-consuming process, especially for large state-action spaces and multi-agent settings. More importantly, the training process can be "unsafe", given that mimicking models' decisions may diverge from that of the original policy models: though fidelity can be measured at the end of the training process, we need to ensure high fidelity *during* training. In conclusion, while the mimic learner can provide explanations about the decisions of an inferred DRL model, it neither allows examining the knowledge accumulated throughout the training process, nor ensures fidelity during the training process.

To deal with these challenges, in this paper we propose a generic interpretable DRL paradigm, where interpretable models are trained in interplay with the original DRL models without requiring these models to be mature: the original model trains the mimicking model and the mimicking model drives the subsequent updates of the original model offering target value estimators during the DRL training process. This is what we call the *online* (i.e. during training and in interplay with other models) training approach. This approach assures fidelity of decisions of the mimicking model, w.r.t. those of the original model. We conjecture that such a paradigm is effective in many DRL methods and can be applied to value based, policy based or actor-critic methods, depending on the model that is mimicked, over discrete or continuous state-action spaces.

To exemplify the proposed paradigm and provide evidence for its applicability, this paper proposes the *e**X**plainable **D**eep **Q**-**N**etwork* (***XDQN***) method, an explainable variation of the well-known DQN [19] method, with the goal to provide inherent explainability of DQN via mimic learning in an online manner. By following an online mode of training the mimicking model, XDQN does not require the existence of a well-trained model to train an interpretable one: The mimic learner is trained and updated while the DQN Q-value model is trained and updated, supporting the maintenance of multiple "snapshots" of the model while it evolves through time, offering interpretability on intermediate models, and insights about the patterns and behaviors that DQN learns during training.

To evaluate the effectiveness of XDQN, this is challenged in complex, real-world multi-agent tasks, where thousands of agents aim to solve airspace congestion problems. As it is shown in other works, the use of independent DQN agents has reached unprecedented performance [13] and therefore such tasks provide a suitable real-world testbed for the proposed method. Agents in this setting are trained via parameter sharing following the centralized training decentralized execution (CTDE) schema.

We summarize the main contributions of this paper below:

- To our knowledge, this work is the first that provides *inherent* interpretability through online mimicking of DRL models, without requiring the existence of a well-trained DRL model: As far as we know, there is not any work that supports this straightforward paradigm for interpretability.
- XDQN exemplifies this paradigm and is proposed as an explainable variation of DQN, in which an interpretable mimic learner is trained online, in interplay with the Q-network of DQN, playing the role of the target Q-network.
- Experimentally, it is shown that XDQN can perform similarly to DQN, demonstrating good play performance and fidelity to DQN decisions in complex real-world multi-agent problems.
- The ability of our method to provide global (policy) and local (in specific circumstances) explanations regarding agents' decisions, also while models are being trained, is demonstrated in a real-world complex setting.

The structure of this article is as follows: Sect. 2 provides background definitions on DRL, deep Q-networks, mimicking approaches, and clarifies the paradigm introduced. Section 3 presents related work. Section 4 exemplifies the proposed paradigm with XDQN, and Sect. 5 provides details on the experimental setting and results, as well as concrete examples of local and global explainability. Section 6 concludes the article.

2 Background

We consider a sequential decision-making setup, in which an agent interacts with an environment E over discrete time steps. At a given timestep, the agent perceives features regarding a state $s_t \in S$, where S is the set of all states in agent's environment (state space). The agent then chooses an action a_t from its repertoire of actions A (i.e., the actions that it can perform in any state), and gets the reward r_t generated by the environment.

The agent's behavior is determined by a policy π, which maps states to a probability distribution over the actions, that is $\pi\colon S \rightarrow P(A)$. Apart from an agent's policy, the environment E may also be stochastic. We model it as a Markov Decision Process (MDP) with a state space S, action space A, an initial state distribution $p(s_1)$, transition dynamics $p(s_{t+1}|s_t, a_t)$ and a reward $r(s_t, a_t, s_{t+1})$, for brevity denoted as r_t.

The agent aims to learn a stochastic policy π^* which drives it to act so as to maximize the expected discounted cumulative reward $G_t = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau$, where $\gamma \in (0, 1)$ is a discount factor.

2.1 Deep Reinforcement Learning with Interpretable Models.

To deal with a high dimensional state space, a policy can be approximated by exploiting a DNN with weight parameters θ. DRL methods learn and exploit different models (e.g. objectives model, value models, models of the environment), which support updating the policy model to fit to the sampled experience generated while interacting with the environment.

In this work we introduce a paradigm for providing DRL methods with inherent interpretability, by replacing closed-box models with interpretable ones. This follows the interpretable box design paradigm specified in [29].

But, how to train interpretable models? Interpretable models can be trained either during or after training the DRL models. Being trained during the training of the DRL models, the interpretable model evolves as the DRL model evolves, and can be used to explain how the training process affects agent's responses. However, this may result to instability and inefficiency of the training process since, interpretable models may aim to reach a moving target and may suffer from high variance. Such pitfalls can be mitigated by means of the "interplay" of interpretable and original models: The interpretable model is trained by specific instances of the DRL model, and its decisions affect subsequent updates of the original model. As an important implication of the online training, the fidelity of the mimicking models with respect to the original DRL models is empirically assured.

Such an online training scheme is followed, for instance, by DRL architectures exploiting a target network (succinctly presented below). A target network provides a stable objective in the learning procedure, and allows a greater coverage of the training data. Target networks, in addition to the benefits they provide in the learning procedure, they can support interpretability of the policy models, given that these can be replaced by interpretable models that are trained with the deep networks in an online manner.

The introduced paradigm for inherently interpretable DRL through online mimicking can be applied to different closed-box DRL models. Here, we exemplify and test this idea to Deep Q-networks, training interpretable Q-value models online with closed-box Q-value models through mimicking. Q-value closed-box and/or interpretable models can be used to extract a DRL policy.

2.2 Deep Q-Networks

Considering that an agent acts under a stochastic policy π, the Q-function (state-action value) of a pair (s, a) is defined as follows

$$Q^{\pi}(s,a) = \mathbb{E}\left[G_t \mid s_t = s, a_t = a, \pi\right] \tag{1}$$

which can also be computed recursively with bootstrapping:

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_t + \gamma \mathbb{E}_{a \sim \pi(s_{t+1})}[Q^{\pi}(s_{t+1}, a)] \mid s_t = s, a_t = a, \pi\right] \tag{2}$$

The Q-function measures the value of choosing a particular action when the agent is in this state. We define the optimal policy π^* under which the agent receives the optimal $Q^*(s,a) = max_{\pi} Q^{\pi}(s,a)$. For a given state s, under the optimal policy π^*, the agent selects action $a = argmax_{a' \in A} Q^*(s, a')$. Therefore, it follows that the optimal Q-function satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_a Q^*(s_{t+1}, a) \mid s_t = s, a_t = a, \pi\right]. \tag{3}$$

In Deep Q-Networks (DQN), to estimate the parameters θ of the Q-values model, at iteration i the expected mean squared loss between the estimated Q-value of a state-action pair and its temporal difference target, produced by a fixed and separate *target* Q-network $Q(s, a; \theta^-)$ with weight parameters θ^-, is minimized. Formally:

$$L_i(\theta_i) = \mathbb{E}\left[Y_i^{DQN} - Q(s, a; \theta)\right], \tag{4}$$

with

$$Y_i^{DQN} = r_t + \gamma \max_{a \in A} Q(s_{t+1}, a; \theta^-) \tag{5}$$

In order to train DQN and estimate θ, we could use the standard Q-learning update algorithm. Nevertheless, the Q-learning estimator performs very poorly in practice. To stabilize the training procedure of DQN, Mnih et al. [19] freezed the parameters, θ^-, of the target Q-network for a fixed number of training iterations while updating the closed Q-network with gradient descent steps with respect to θ.

The direct application of the online mimicking approach in DQN uses an interpretable target DQN model to mimic the online Q-network, and thus, the decisions of the original policy model.

In addition to the target network, during the learning process, DQN uses an experience replay buffer [19], which is an accumulative dataset, D_t, of state transition samples - in the form of (s, a, r, s') - from past episodes. In a training step, instead of only using the current state transition, the Q-Network is trained by sampling mini-batches of past transitions from D uniformly, at random. Therefore, the loss can be written as follows:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)}\left[(Y_i^{DQN} - Q(s, a; \theta))^2\right]. \tag{6}$$

As it is well known, the main advantage of using an experience replay buffer is that uniform sampling reduces the correlation among the samples used for training the Q-network. The replay buffer improves data efficiency through reusing the experience samples in multiple training steps. Instead of sampling minibatches of past transitions uniformly from the experience replay buffer, a further improvement over DQN results from using a prioritized experience replay buffer [24]. This aims at increasing the probability of sampling those past transitions from the experience replay that are expected to be more useful in terms of absolute temporal difference error.

3 Related Work

Explainability in Deep Reinforcement Learning (DRL) is an emergent area whose necessity is related to the fact that DRL agents solve sequential tasks, acting in the real-world, in operational settings where safety, criticality of decisions and the necessity for transparency (i.e. explainability with respect to real-world pragmatic constraints [29]) is the norm. However, DRL methods use closed-boxes whose functionality is intertwined and are not interpretable. This may hinder DRL methods explainability. In this paper we address this problem by proposing an interpretable DRL method comprising two models which are trained jointly in an online manner: An interpretable mimicking model and a deep model. The later offers training samples to the mimicking one and the former interpretable model offers target action values for the other to improve its predictions. At the end of the training process, the mimicking model has the capacity to provide high-fidelity interpretations to the decisions of the deep model and thus, it can replace the deep model. This proposal is according to the interpretable box design paradigm, which follows the conjecture (stated for instance in [22]) that there is high probability that decisions of closed-boxes can be approximated by well designed interpretable models.

There are many proposals for interpreting Deep Neural Networks (DNNs) through mimicking approaches. These approaches differ in several dimensions: (a) the targeted representation (e.g., decision trees in DecText [5], logistic model trees (LMTs) in reference [8], or Gradient Boosting Trees in reference [6]), (b) to the different splitting rules used towards learning a comprehensive representation, (c) to the actual method used for building the interpretable model (e.g., [8] uses the LogiBoost method, reference [5] proposes the DecText method, while the approach proposed in reference [6] proposes a pipeline with an external classifier, (d) on the way of generating samples to expand the training dataset. These methods can be used towards interpreting constituent individual DRL models employing DNNs. Distillation could be another option [23], but it typically requires mature DRL models: Online distillation of models, as far as we know, has not been explored. The interested reader is encouraged to read a thorough review on these methods provided in [3,12,20,22].

Recent work on mimic learning [6,16] has shown that rule-based models, like decision trees, or shallow feed-forward neural networks can mimic a not linear function inferred by a DNN with millions of parameters. The goals here is to train a mimic model with efficiency, resulting into a high performance model, which takes decisions in high-fidelity with respect to the decisions of the original model.

For DRL, authors in [16] introduce Linear Model U-trees (LMUTs) to approximate predictions for DRL agents. An LMUT is learned by an algorithm that is well-suited for an active play setting. The use of LMUTs is compared against using CART, M5 with regression tree, Fast Incremental Model Tree (FIMT) and with Adaptive Filters (FIMT-AF). The use of decision trees as interpretable policy models trained through mimicking has been also investigated in [18], in conjunction to using a causal model representing agent's objectives and opportunity chains. However, the decision tree in this work is used to infer the effects of actions approximating the causal model of the environment. Similarly to what we do here, the decision tree policy model is trained concurrently with the RL policy model, assuming a model-free RL algorithm and exploiting state-action samples using an experience replay buffer. In [7] authors illustrate how Soft Decision Trees (SDT) [10] can be used in spatial settings as interpretable policy models. SDT are hybrid classification models of binary trees of predetermined depth, and neural networks. However their inherent interpretability is questioned given their structure. Other approaches train interpretable models other than trees, such as the Abstracted Policy Graphs (APGs) proposed in [28], assuming a well-trained policy model. APGs concisely summarize policies, so that individual decisions can be explained in the context of expected future transitions.

In contrast to the above mentioned approaches, the proposed paradigm, exemplified by means of the proposed XDQN algorithm, can be applied to any setting with arbitrary state features, where the interpretable model is trained jointly to the deep model through online mimicking.

It is worth noting that, instead of the Gradient Boosting Regressors mimic learner for XDQN, we also tested naturally interpretable Linear Trees (such as LMUTs [16]); i.e. decision trees with linear models in their leaves). However, such approaches demonstrated quite low play performance with very low fidelity in the real-world complex experimental setting considered.

Regarding mimicking the Q-function of a DRL model, two known settings are the experience training and the active play settings.

In the experience training setting [6,16], all the state-action pairs $\langle s, a\rangle$ of a DRL training process are collected in a time horizon T. Then, to obtain the corresponding Q-values, these pairs are provided as input into a DRL model. The final set of samples $\{(\langle s_1, a_1\rangle, Q_1), ...(\langle s_T, a_T\rangle, Q_T)\}$ is used as the experience training dataset. The main problem with the experience training is that suboptimal samples are collected through training, making it more difficult for a learner to mimic the behavior of the DRL model.

Active play [16] uses a mature DRL model to generate samples to construct the training dataset of an active mimic learner. The training data is collected in an online manner through queries, in which the active learner selects the actions, given the states, and the mature DRL model provides the estimated Q-values. These Q-values are then used to update the active learner's parameters on minibatches of the collected dataset. While the pitfall of suboptimal samples is addressed here, active play cannot eliminate the need for generating new trajectories to train the mimic models, which can be computationally prohibitive for real-world controllers.

Rather than following an experience or an active play training scheme, in this paper we use online training, that collects samples generated from a DRL model in a horizon of training timesteps, without requiring these samples to be generated from a mature DRL model. Samples are gathered from intermediate DRL models, during the DRL training process.

As for the provision of explanations, we opted for features' contributions to the Q-values, in a rather aggregated way, using the residue of each Gradient Boosting Regressor node, as done in [9]. This approach, as shown in [9], reports advantages over using well known feature importance calculation methods, avoiding linearity assumptions made by LIME [21] and bias in areas where features have high variance. It also avoids taking all tree paths into account in case of outliers, as done, for instance, by SHAP [17].

4 Explainable DRL Through Online Mimicking: The Deep Q-Network (XDQN) Example

To demonstrate the inherent interpretability of deep Q-learning through online mimicking of the Q-network, we propose *eXplainable Deep Q-Network* (*XDQN*)[1], which is an explainable variation of DQN [19]. XDQN aims at inferring a mimic learner, trained in an online manner, substituting the target Q-network of DQN.

Formally, let θ be the parameters of the Q-network and $\tilde{\theta}$ be the parameters of the mimic learner. In XDQN, the mimic learner estimates the state-action value function and selects the best action for the next state playing the role of the XDQN target:

$$Y_i^{XDQN} = r_t + \gamma \max_{a \in A} Q\left(s_{t+1}, a; \tilde{\theta}\right) \tag{7}$$

Similar to DQN, $\tilde{\theta}$ are updated every T_u number of timesteps. The full training procedure of XDQN is presented in Algorithm 1.

In contrast to DQN in which the parameters θ of the Q-network are simply copied to the target Q-network, here we perform mimic learning on $Q(s, a, \theta)$ (steps 17–20). To update $\tilde{\theta}$ we train the mimic learner on minibatches of the experience replay buffer B by minimizing the Mean Squared Error (MSE) loss function using $Q(s, a, \theta)$ to estimate the soft labels (Q-values) of the state-action pairs in the minibatches. The problem for optimizing $\tilde{\theta}$ at each update can be written as:

$$\min_{\tilde{\theta}} \mathbb{E}_{(s,a)\sim B}\left[\left(Q(s, a; \tilde{\theta}) - Q(s, a; \theta)\right)^2\right] \tag{8}$$

[1] The implementation code will be made available in the final version of the manuscript.

Algorithm 1 eXplainable Deep Q-Network (XDQN)

```
1: Initialize replay buffer $B$ with capacity N
2: Initialize $\theta$ and $\tilde{\theta}$
3: Initialize timestep count $c = 0$
4: for episode 1, M do
5:     Augment $c = c + 1$
6:     Initialize state $s_1$
7:     With probability $\epsilon$ select a random action $a_t$, otherwise $a_t = argmax_a Q(s_t, a; \theta)$
8:     Execute action $a_t$ and observe next state $s_{t+1}$ and reward $r_t$
9:     Store transition $(s_t, a_t, s_{t+1}, r_t)$ in $B$
10:    Sample a minibatch of transitions $(s_i, a_i, s_{i+1}, r_i)$ from $B$
11:    if $s_{i+1}$ not terminal then
12:        Set $Y_i^{XDQN} = r_i + \gamma \max_{a \in A} Q\left(s_{i+1}, a; \tilde{\theta}\right)$
13:    else
14:        Set $Y_i^{XDQN} = r_i$
15:    end if
16:    Perform a gradient descent step on $\left(Y_i^{XDQN} - Q(s_i, a_i; \theta)\right)^2$ w.r.t. $\theta$
17:    if $c$ mod $T_u = 0$ then
18:        Initialize $\tilde{\theta}$
19:        Sample a minibatch of transitions $(s_i, a_i, s_{i+1}, r_i)$ from $B$ that were stored
   at most $c - K$ timesteps before
20:        Perform mimic learning update on $\left(Q(s, a; \tilde{\theta}) - Q(s, a; \theta)\right)^2$ w.r.t $\tilde{\theta}$
21:    end if
22: end for
```

where, B is the prioritized experience replay buffer [24], as described in Sect. 2. Similarly to active play, when updating $\tilde{\theta}$, to ensure that the samples in minibatches provide up-to-date target values with respect to θ, we use records from the replay buffer that were stored during the K latest training steps.

It is worth noting that the hyperparameter K plays a similar role as the discounted factor γ plays for future rewards, but from the mimic learner's perspective. Building upon the experience training and active play schemes, the online training scheme leverages the benefits of both of them, aiming to minimize the required sample complexity for training the mimic model without simulating new trajectories of a mature DRL model. In particular, hyperparameter K manages the trade-off between experience training and active play in XDQN. If K is large, the mimic model learns from samples that may have been collected through more suboptimal instances of θ; deploying however data-augmented versions of Q-value. On the other hand, if K is small, it learns from the most recent instances of θ; making use of up-to-date Q-values. Nevertheless, opting for very small values of K could lead to less stable mimic training, due to the smaller number of minibatches that can be produced for updating $\tilde{\theta}$, while using large K can result in a very slow training process.

From all the above, we note that θ (Q-network) and $\tilde{\theta}$ (mimic learner) are highly dependent. To update θ, Q-network uses the mimic learner model with $\tilde{\theta}$

to compute the target soft labels (target Q-values), while to update $\tilde{\theta}$ the mimic learner uses the original Q-network with parameters θ to compute the respective target soft labels (Q-values). It is conjectured that through this dependency the interpretable target model converges to Q-values that are close to the values of the Q-network, and thus to nearly the same policy, which is an inherent feature of the DQN algorithm.

Since XDQN produces different instances of $\tilde{\theta}$ throughout training, it can eventually output multiple interpretable mimic learner models (up to the number of $\tilde{\theta}$ updates), with each one of them corresponding to a different training timestep. Since all these mimic learner instances are interpretable models, XDQN can also provide information on how the Q-network evolves towards solving the target task.

Finally, after Q-network (θ) and mimic learner ($\tilde{\theta}$) have been trained, we can discard the closed-box Q-network and use the mimic learner model as the controller. Therefore, in testing, given a state, the interpretable mimic learner selects the action with the highest Q-value, being able to also provide explainability.

5 Experimental Setup

This section demonstrates the effectiveness of XDQN through experiments in real-world settings pertaining to the demand-capacity balancing (DCB) problem of air traffic management (ATM) domain. XDQN uses a Gradient Boosting Regressor (GBR) [30] mimic learner, whose boosting ability supports effective learning by exploiting instances generated by the deep Q-network. We opted for GBR, as it usually results into robust and accurate models compared to other decision tree - based models.

Although the boosting structure of GBR makes it very difficult to provide explainability, following the work in [9] we are able to measure the contribution of state features to the predicted Q-values. In so doing, the mimic learner is expected to give local and global explanations on its decisions.

Overall, we are interested in demonstrating the proposed paradigm and show the importance of online training of mimic interpretable models. In so doing, the performance of XDQN is compared to that of DQN in complex real-world DCB problems.

5.1 Real-World Demand-Capacity Problem Setting

The current ATM system is based on time-based operations resulting in DCB [15] problems. To solve the DCB issues at the pre-tactical stage of operations, the ATM system opts for methods that generate delays and costs for the entire system. In ATM, the airspace consists of a set of 3D sectors where each one has a specific capacity. This is the number of flights that cross the sector during a specific period (e.g. of 20 min). The challenge of dealing with the DCB problem is to reduce the number of congestion cases (DCB issues, or *hotspots*), where the demand of airspace use exceeds its capacity, with small delays to an - as much as possible - low number of flights.

Recent work has transformed the DCB challenge to a multi-agent RL problem by formulating the setting as a multi-agent MDP [15]. We follow the work and the experimental setup of [13–15, 25, 26] and encourage the reader to see the problem formulation [15] in details. In this setting, we consider a society of agents, where each agent is a flight (related to a specific aircraft) that needs to coordinate its decisions regarding minutes of delay to be added in its existing delay, so as to resolve hotspots that occur, together with the other society agents. Agents' local states comprise 81 state variables related to: (a) the existing minutes delay, (b) the number of hotspots in which the agent is involved in, (c) the sectors that it crosses, (d) the minutes that the agent is within each sector it crosses, (e) the periods in which the agent joins in hotspots in sectors, and (f) the minute of the day that the agent takes off. The tuple containing all agents' local states is the joint global state. Q-learning [27] agents have been shown to achieve remarkable performance on this task [13]. In our experiments, all agents share parameters and replay buffer, but act independently.

A DCB scenario comprises multiple flights crossing various airspace sectors in a time horizon of 24 h. This time horizon is segregated into simulation time steps. At each simulation time step (equal to 10 min of real time), given only the local state, each agent selects an action which is related to its preference to add ground delay regulating its flight, in order to resolve hotspots in which it participates. The set of local actions for each agent contains $|\text{maxDelay} + 1|$ actions, at each simulation time step. We use $\text{maxDelay} = 10$. The joint (global) action is a tuple of local actions selected by the agents. Similarly, we consider local rewards and joint (global) rewards. The local reward is related to the cost per minute within a hotspot, the total duration of the flight (agent) in hotspots, as well as to the delay that a flight has accumulated up to the simulation timestep [13].

5.2 Evaluation Metrics and Methods

For the evaluation of the proposed method, first, we make use of two known evaluation metrics: (a) *play performance* [16] of the deep Q-network, and (b) *fidelity* [29] of the mimic learner. Play performance measures how well the deep Q-network performs with the mimic learner estimating its temporal difference targets, while fidelity measures how well the mimic learner matches the predictions of the deep Q-network.

As far as play performance is concerned, in comparison with results reported in [13], we aim at minimizing the number of *hotspots*, the *average delay per flight* and the number of *delayed flights*. As for fidelity, we use two metric scores: (a) the *mean absolute error (MAE) of predicted Q-values* and (b) the *mimicking accuracy* score. Given a minibatch of states, we calculate the MAE of this minibatch for any action as the mean absolute difference between the Q-values estimated by the mimic learner and the Q-values estimated by the deep Q-network for that action. More formally, for a minibatch of states D_s, the MAE_i of action a_i is denoted as:

$$MAE_i = \frac{1}{|D_s|} \sum_{s \in D_s} |Q(s, a_i; \tilde{\theta}) - Q(s, a_i; \theta)| \tag{9}$$

It is worth noting that minimizing the MAE of the mimic learner is very important for training XDQN. Indeed, training a mimic model to provide the target Q-values, large MAEs can lead the deep Q-network to overestimate bad states and understimate the good ones, and thus, find very diverging policies that completely fail to solve the task.

Given a minibatch of samples, mimicking accuracy measures the percentage of the predictions of the two models that agree with each other, considering that models select the action with the highest estimated Q-value.

Second, we illustrate XDQN's *local* and *global* interpretability. We focus on providing aggregated interpretations, focusing on the contribution of state features to local decisions and to the overall policy: This, as suggested by ATM operators, is beneficial towards understanding decisions, helping them to increase their confidence to the solutions proposed, and mastering the inherent complexity in such a multi-agent setting, as solutions may be due to complex phenomena that are hard to be traced [13]. Specifically, in this work, local explainability measures state features' importance on a specific instance (i.e. a single state-action pair), demonstrating which features contribute to the selection of a particular action over the other available ones. Global explainability aggregates features' importance on particular action selections over many different instances, to explain the overall policy of the mimic learner.

Finally, the evolution of the DRL model throughout the training process is demonstrated through GBR interpretability.

5.3 Experimental Scenarios and Settings

Experiments were conducted on three in total scenarios. Each of these scenarios corresponds to a date in 2019 with heavy traffic in the Spanish airspace. In particular, the date scenarios, on which we assess our models, are 20190705, 20190708 and 20190714. However, to bootstrap the training process we utilize a deep Q-network pre-trained in various scenarios, also including 20190705 and 20190708, as it is done in [15]. In the training process, the deep Q-network is further trained according to the method we propose. The experimental scenarios were selected based on the number of hotspots and the average delay per flight generated in the ATM system within the duration of the day, which shows the difficulty of the scenario. Table 1 presents information on the three experimental scenarios. In particular, the flights column indicates the total number of flights (represented by agents) during the specific day. The initial hotspots column indicates the number of hotspots appearing in the initial state of the scenario. The flights in hotspots column indicates the number of flights in at least one of the initial hotspots. Note that all three scenarios display populations of agents (flights) of similar size, within busy summer days. For each scenario we ran five separate experiments and average results.

Table 1. The three experimental scenarios.

Scenario	Flights	Initial Hotspots	Flights in Hotspots
20190705	6676	100	2074
20190708	6581	79	1567
20190714	6773	92	2004

The implementation of XDQN utilizes a deep multilayer perceptron as the deep Q-network. The maximum depth of the Gradient Boosting Regressor is set equal to 45 and the number of minimum samples for a split equal to 20. We use the mean squared error as the splitting criterion. To train a single decision tree for all different actions, a non binary splitting rule of the root is used, based on the action size of the task, so that the state-action pairs sharing the same action match the same subtree of the splitting root. XDQN uses an ϵ-greedy policy, which at the start of exploration has ϵ equal to 0.9 decaying by 0.01 every 15 episodes until reaching the minimum of 0.04 during exploitation. The total number of episodes are set to 1600 and the update target frequency is set to 9 episodes. The memory capacity of the experience replay for the online training of the mimic learner, i.e. the hyperparameter K, is set equal to the 1/20 of the product of three other hyperparameters, namely the total number of timesteps per episode (set to 1440), the update target frequency (set to 9) and the number of agents (set to 7000). Thus, K is set to 4536000 steps.

5.4 Evaluation of Play Performance

Table 2. Comparison of performance of XDQN and DQN on the three experimental ATM scenarios (*FH*: Final Hotspots, *AD*: Average Delay, *DF*: Delayed Flights).

Scenario	DQN			XDQN		
	FH	AD	DF	FH	AD	DF
20190705	38.4	13.04	1556.5	39.0	13.19	1618.54
20190708	4.6	11.4	1387.2	6.0	11.73	1331.58
20190714	4.8	10.72	1645.2	7.0	13.46	1849.49

Table 2 demonstrates the play performance of DQN and XDQN on the three experimental scenarios. The final hotspots column indicates the number of unresolved hotspots in the final state: It must be noted that these hotspots may have emerged due to delays assigned to flights and may be different than the hotspots at the beginning of each scenario. The average delay per flight column shows the total minutes of delay imposed to all flights (when the delay is more that 4 min, according to operational practice), divided by the number of flights in the specific scenario. The delayed flights column indicates the number of flights affected by more than 4 min of delay.

We observe that XDQN performs similarly to DQN in all three evaluated metric scores, reducing considerably the number of hotspots in the scenarios, and assigning delays to the same proportion of flights. In particular, while DQN slightly outperforms XDQN in terms of the final hotspots and average delay in all three scenarios, XDQN decreases the number of the delayed flights in one scenario, while it demonstrates competitive performance on the others. This demonstrates the ability of XDQN to provide qualitative solutions while offering transparency in decision making, in contrast to DQN, which offers slightly better solutions that, however, are difficult (or even - due to their complexity - impossible) to be understood by humans [13].

5.5 Evaluation of Fidelity

As discussed in Subsect. 4.2, for the fidelity evaluation we measure the mean absolute error (MAE) between models' predicted Q-values and the mimicking accuracy score of the interpretable model. Given the DCB experimental scenarios, we train three different mimic models; namely X0705, X0708 and X0714. Table 3 reports the average MAE for each decided action over all mimic learning updates. We observe that all errors are very small, given that in testing, the absolute Q-values hovered around 200. This is very important for stabilizing the training process of XDQN, since mimic-model Q-value predictions should be ideally equal to the ones generated by the deep Q-network.

Table 3. Average Mean Absolute Errors (MAE) of the mimic model over three updates.

Action (Delay Option)	XDQN mimic model update		
	X0705	X0708	X0714
0	0.279	0.237	0.291
1	1.766	1.971	1.942
2	0.910	0.928	1.002
3	0.575	0.661	0.640
4	0.639	0.748	0.725
5	1.893	2.096	2.121
6	1.590	1.766	1.715
7	1.610	1.816	1.733
8	0.449	0.514	0.497
9	0.740	0.849	0.823
10	1.292	1.525	1.461

To further assess the fidelity of XDQN mimic learner, Table 4 illustrates the average mimicking accuracy scores over all mimic learning updates and the corresponding accuracy scores of the final mimic model. Since a Gradient Boosting Regressor mimic learner is a boosting algorithm, it produces sequential decision trees that can successfully separate the state space and approximate well the predictions of the deep Q-network function. We observe that the mimic learner and the deep Q-network agree with each other to a very good extent; namely from approximately 81% to 92%, including the final mimic model. Therefore, we expect the mimic learner to be able to accumulate the knowledge from the deep Q-network with high fidelity.

Table 4. The accuracy scores of the mimic models.

Scenario	mimicking accuracy (%) (average over training steps)	mimicking accuracy (%) (final model)
20190705	88.45	87.53
20190708	81.89	82.39
20190714	90.88	92.63

5.6 Local and Global Explainability

In the DCB setting, it is important for the human operator to understand how the system reaches decisions on regulations (i.e. assignment of delays to flights): as already pointed out, this should be done at a level of abstraction that would allow operators to understand the rationale behind decisions towards increasing their confidence to the solutions proposed, mastering the inherent complexity of the setting, and further tune solutions when necessary, without producing side-effects that will increase congestion and traffic. Therefore, operators are mainly interested in receiving succinct explanations about which state features contribute to the selection of delay actions (i.e. actions larger than 1 min) over the no-delay action (i.e. action equal to 0 min).

First, we demonstrate the ability of the mimic learner to provide local explainability. As already said, local explainability involves showing which state features contribute to the selection of a particular action over the other available ones in a specific state. To this aim, for any pair of actions - a_1 and a_2 - we calculate the differences of feature contributions in selecting a_1 and a_2 in a single state. To highlight the most significant differences, we focus only on those features whose absolute differences are above a threshold. Empirically, we set this threshold equal to 0.5. Figure 1 illustrates local explainability on a given state in which action "2" was selected: It provides the differences of feature contributions to the estimation of Q-values when selecting action "0" against selecting action "2" (denoted by "0–2"). We observe that the features that contributed more to the selection of the delay action "2" were those with index 32 (i.e. The sector in which the last hotspot occurs), 2 (i.e. the sector in which the first hotpot occurs)

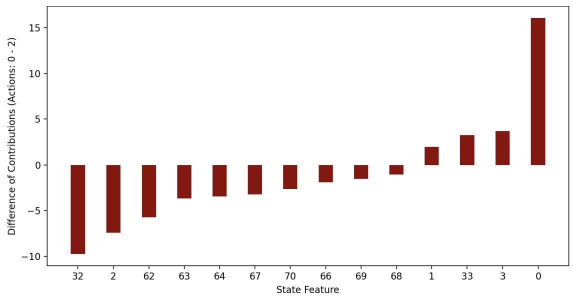


Fig. 1. Illustration of significant differences of feature contributions in selecting action "0" and action "2" in a single state, in which action "2" was selected. Positive differences mean that the respective state features have a greater contribution to Q-value when action "0" is selected, rather than when action "2" is selected. Negative differences have the opposite meaning.

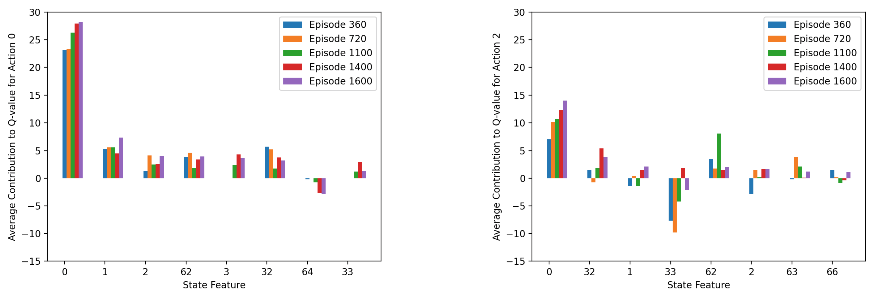
Table 5. The most significant state features in terms of average contribution difference (ACD) in selecting the no-delay action versus a delay action. A positive ACD means that the corresponding state feature on average contributes more to the selection of the no-delay action "0". On the contrary, a negative ACD means that the corresponding state feature on average contributes to the selection of a delay action "1–10".

Feature Index	Feature Meaning	ACD
0	Delay the flight has accumulated up to this point	Positive
1	Total number of hotspots the flight participates in	Positive
2	The sector in which the first hotspot the flight participates occurs	Negative
3	The sector in which the second hotspot the flight participates occurs	Positive
32	The sector in which the last hotspot the flight participates occurs	Negative
62	The minutes the flight remains in the last sector it crosses	Negative
63	The minute of day the flight takes off given the delay (CTOT)	Negative
64	The minutes the flight remains in the first sector it crosses	Negative
68	The minutes the flight remains in the fifth sector it crosses	Negative

and 62 (i.e. the minutes that the flight spends crossing the last sector). Similarly to the explanations provided in [29], the arguments in favor of receiving additional minutes of delay concern the hotspots in which the flight participates, as well as the duration of the time span in which the flight crosses congested sectors (and mainly the first sector), as well as the delay that the flight has accumulated up to this point (if this is low). On the contrary, the arguments against receiving delay concern the delay that the flight has accumulated up to this point (if this is somehow high), and the small duration of the time span the flight spends in congested sectors.

Finally, we demonstrate XDQN's global explainability by aggregating the importance of features on particular action selections over many different state-action instances. In particular, we are interested in measuring the state feature contributions to the selection of delay actions (i.e. actions in the range $[1, 10]$) over the no-delay action (i.e. action “0”) in the overall policy. To this aim, all pairs of actions, with one action always being the no-delay action and the other one being a delay action, are considered. For each such pair, the differences of feature contributions to estimating the actions’ Q-values over many different state-action instances are averaged: This results into features’ Average Contribution Difference (ACD). Table 5 shows the most significant state features in terms of ACD in selecting the no-delay action versus a delay action. The most significant features, are (currently set to eight) features with the highest absolute ACD, for each action in the range $[1, 10]$) over the no-delay action. We observe that features with index 0, 3 contribute more to the selection of the no-delay action. On the contrary, features with indexes 32, 2, 62 contribute more to the selection of a delay action.

Last but not least, we demonstrate how global explainability evolves through the training process, addressing the question of how a DRL model learns to solve the target task. To this aim, features’ contribution to selecting an action are averaged. This results into Average Features’ Contribution (AFC). The (currently set to eight) most significant features, i.e. those with the highest absolute AFC, are those considered in explanations. Then the AFC to predicting the Q-value of a selected action at different training episodes is provided for the most significant features: Fig. 2 illustrates the evolution of global explainability for selecting the no-delay action (left) and a delay action (right) through 5 representative training episodes (360th, 720th, 1100th, 1400th and 1600th), in terms of eight features with the highest AFC values in the final model (episode 1600). We observe that for both evaluated actions most of the features show an increasing/decreasing trend in their average contribution to Q-value over time, such as those with indices 0, 1 and 63. It is worth noting that although the features 0 and 1 have



(a) (b)

Fig. 2. Illustration of the evolution of features’ contributions for selecting the no-delay action (“0”) and a delay one (“2”) through 5 representative training episodes (360th, 720th, 1100th, 1400th and 1600th) in terms of average feature contribution (AFC) to Q-value for the eight features with highest absolute AFC values in the final model (episode 1600) in the selection of the aforementioned actions.

been highlighted as the most significant for the selection of the no-delay action, they have also significant but less contribution to selecting a delay action, as well.

6 Conclusion and Future Work

This work shows how interpretable models can be trained through online mimicking and substitute closed-box DRL models, providing inherent DRL interpretability, or be used for the provision of DRL methods interpretability. This generic paradigm follows the interpretable box design paradigm for explainable DRL and is exemplified by means of an explainable DQN (XDQN) method, where the target model has been substituted by an interpretable model that provides interpretations regarding the importance of state features in decisions. XDQN, utilizing a Gradient Boosting Regressor as the mimic learner, has been evaluated in challenging multi-agent demand-capacity balancing problems pertaining to air traffic management. Experimentally, we have shown that the XDQN method performs on a par with DQN in terms of play performance, whereas demonstrating high fidelity and global/local interpretability.

Overall, this work provides evidence for the capacity of the proposed paradigm to provide DRL methods with inherent interpretability, with high play performance and high-fidelity to the decisions of original DRL models, through online training of interpretable mimicking models.

Further work is necessary to explore how this paradigm fits into different types of DRL architectures, utilizing interpretable models that are trained to mimic different DRL closed-box models. In this line of research, we need to benchmark our methodology, utilizing state-of-the-art DRL in various experimental settings.

Regarding XDQN, further work is to design, evaluate and compare various explainable mimic models that can effectively substitute the target Q-Network.

Acknowledgements. This work has been supported by the TAPAS H2020-SESAR2019-2 Project (GA number 892358) Towards an Automated and exPlainable ATM System and it is partially supported by the University of Piraeus Research Center.

References

1. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, vol. 27. Curran Associates, Inc. (2014). https://proceedings.neurips.cc/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf
2. Bastani, O., Pu, Y., Solar-Lezama, A.: Verifiable reinforcement learning via policy extraction. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS 2018, pp. 2499–2509. Curran Associates Inc., Red Hook, NY, USA (2018)

3. Belle, V., Papantonis, I.: Principles and practice of explainable machine learning. Front. Big Data **4**, 39 (2021)
4. Boz, O.: Extracting decision trees from trained neural networks. In: KDD 2002, pp. 456–461. Association for Computing Machinery, New York, NY, USA (2002). https://doi.org/10.1145/775047.775113
5. Boz, O.: Extracting decision trees from trained neural networks. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 456–461 (2002)
6. Che, Z., Purushotham, S., Khemani, R., Liu, Y.: Interpretable deep models for ICU outcome prediction. In: AMIA Annual Symposium Proceedings 2016, pp. 371–380, February 2017
7. Coppens, Y., et al.: Distilling deep reinforcement learning policies in soft decision trees. In: Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence, pp. 1–6 (2019)
8. Dancey, D., Bandar, Z.A., McLean, D.: Logistic model tree extraction from artificial neural networks. IEEE Trans. Syst. Man Cybern. Part B (Cybern.) **37**(4), 794–802 (2007)
9. Delgado-Panadero, Á., Hernández-Lorca, B., García-Ordás, M.T., Benítez-Andrades, J.A.: Implementing local-explainability in gradient boosting trees: feature contribution. Inf. Sci. **589**, 199–212 (2022). https://doi.org/10.1016/j.ins.2021.12.111, https://www.sciencedirect.com/science/article/pii/S0020025521013323
10. Frosst, N., Hinton, G.: Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784 (2017)
11. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3389–3396 (2017). https://doi.org/10.1109/ICRA.2017.7989385
12. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. (CSUR) **51**(5), 1–42 (2018)
13. Kravaris, T., et al.: Explaining deep reinforcement learning decisions in complex multiagent settings: towards enabling automation in air traffic flow management. Appl. Intell. (Dordrecht, Netherlands) 53, 4063–4098 (2022)
14. Kravaris, T., et al.: Resolving congestions in the air traffic management domain via multiagent reinforcement learning methods. arXiv:abs/1912.06860 (2019)
15. Kravaris, T., Vouros, G.A., Spatharis, C., Blekas, K., Chalkiadakis, G., Garcia, J.M.C.: Learning policies for resolving demand-capacity imbalances during pre-tactical air traffic management. In: Berndt, J.O., Petta, P., Unland, R. (eds.) MATES 2017. LNCS (LNAI), vol. 10413, pp. 238–255. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64798-2_15
16. Liu, G., Schulte, O., Zhu, W., Li, Q.: Toward interpretable deep reinforcement learning with linear model U-Trees. In: Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., Ifrim, G. (eds.) ECML PKDD 2018, Part II. LNCS (LNAI), vol. 11052, pp. 414–429. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-10928-8_25
17. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in Neural Information Processing Systems, vol. 30 (2017)
18. Madumal, P., Miller, T., Sonenberg, L., Vetere, F.: Explainable reinforcement learning through a causal lens. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 2493–2500 (2020)

19. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
20. Murdoch, W.J., Singh, C., Kumbier, K., Abbasi-Asl, R., Yu, B.: Definitions, methods, and applications in interpretable machine learning. Proc. Nat. Acad. Sci. **116**(44), 22071–22080 (2019)
21. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should i trust you?": Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, pp. 1135–1144. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939778
22. Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L., Zhong, C.: Interpretable machine learning: fundamental principles and 10 grand challenges (2021). https://doi.org/10.48550/ARXIV.2103.11251, arXiv:2103.11251
23. Rusu, A.A., et al.: Policy distillation (2015). https://doi.org/10.48550/ARXIV.1511.06295, arXiv:1511.06295
24. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay (2015). https://doi.org/10.48550/ARXIV.1511.05952, arXiv:1511.05952
25. Spatharis, C., Bastas, A., Kravaris, T., Blekas, K., Vouros, G., Cordero Garcia, J.: Hierarchical multiagent reinforcement learning schemes for air traffic management. Neural Comput. Appl. **35**, 147–159 (2021). https://doi.org/10.1007/s00521-021-05748-7
26. Spatharis, C., et al.: Multiagent reinforcement learning methods to resolve demand capacity balance problems. In: Proceedings of the 10th Hellenic Conference on Artificial Intelligence, SETN 2018. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3200947.3201010
27. Tan, M.: Multi-agent reinforcement learning: Independent versus cooperative agents. In: ICML (1993)
28. Topin, N., Veloso, M.: Generation of policy-level explanations for reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 2514–2521 (2019)
29. Vouros, G.A.: Explainable deep reinforcement learning: state of the art and challenges. ACM Comput. Surv. **55**, 1–39 (2022). https://doi.org/10.1145/3527448,just Accepted
30. Zemel, R.S., Pitassi, T.: A gradient-based boosting algorithm for regression problems. In: Proceedings of the 13th International Conference on Neural Information Processing Systems, NIPS 2000, pp. 675–681. MIT Press, Cambridge, MA, USA (2000)
31. Zhao, X., et al.: DEAR: deep reinforcement learning for online advertising impression in recommender systems. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35(1), pp. 750–758, May 2021. https://ojs.aaai.org/index.php/AAAI/article/view/16156