University of Groningen · Master's Thesis in Artificial Intelligence

Accelerated Learning and Potential Explainability Gains Through Object-Based Deep Reinforcement Learning

Niels de Jong
Faculty of Science and Engineering,
University of Groningen

Dr. Matthia Sabatelli
Faculty of Science and Engingeering,
University of Groningen

7 July 2022

In the last two decades, artificial intelligence (AI) has seen considerable progress due to advances in deep learning (DL). However, the challenges of data-hungriness and model explainability seem to persist in DL's most popular branch, namely supervised DL. Hence, researchers have recently argued to shift attention towards non-supervised DL, such as deep reinforcement learning (DRL). Within DRL, a promising and relatively new avenue involves the use of high-level features ('objects') in models, giving rise to object-based DRL. Motivated by recent results, we investigate whether object-level state representations accelerate learning in DRL, and tentatively attempt to answer whether they make DRL methods more explainable. We do so by conducting three experiments in which we compare, against non-object baselines, the average undiscounted return curves of two existing and one proposed object-based DRL method, while also extracting object saliency maps (OSMs). Results indicate that learning may indeed be accelerated, provided that a sufficiently effective DRL method is chosen and that the representations are presented consistently. Further, using the collected OSMs, we present a critical discussion of the potential use of the OSM as an object-based explainability tool for DRL methods, and suggest how this possible use may be evaluated in future research.

Keywords: Deep reinforcement learning (DRL) · Object-based DRL · Sampling efficiency · Model explainability

Introduction

The field of artificial intelligence (AI) has progressed considerably since its inception at the Dartmouth workshop in 1956 ((Russell & Norvig, 2010), p. 17). Throughout this historic progression, the field may be viewed as oscillating between two contrasting paradigms. On the one hand we have symbolic AI, which has a top-down approach to intelligence. Its emphasis lies on generation, representation, and inference of high-level symbols, such as logical statements. In direct contrast to symbolic AI we have connectionist and situated or embodied AI, which instead work from the bottom up. These methods are relatively numerical and more biologically inspired, and seek to obtain intelligent behaviour from direct, low-level input-output mappings or interactions with the environment (Mira, 2008). During the 1940s, when AI was not yet a stand-alone research area, but instead still a component of cybernetics (Goodfellow, Bengio, & Courville, 2016, pp. 12–17), the field was situated largely within the second paradigm. Then, from the Dartmouth conference until the mid-1980s, researchers shifted their attention towards symbolic AI, after which, from the 1990s onward, the connectionist paradigm took on prominence again, albeit now more familiarly under the name of 'artificial neural networks' (ANNS; Mira, 2008). From around 2006 and onward, deep (artificial) neural networks (DNNS) became viable to train due to broader access to large datasets and advances in computer hardware, perhaps most notably GPUS (Goodfellow et al., 2016). Especially since 2012 (Krizhevsky, Sutskever, & Hinton, 2012), research in AI once again appears firmly located in the connectionist paradigm.

The recent successes achieved by deep learning (DL, the field that studies DNNS) are arguably caused by two primary reasons. The first reason is possibly the most apparent: deep learning methods have enabled advancements in for instance computer vision (Krizhevsky et al., 2012), speech recognition (Mohamed, Dahl, & Hinton, 2012), and natural language processing (Vaswani et al., 2017) because of their unprecedented

Dr. Sabatelli supervises de Jong's project. Correspondence concerning this thesis should be addressed to Niels de Jong, University of Groningen, Broerstraat 5, 9712 CP, Groningen. Email should be sent to n.a.de.jong@student.rug.nl.

capability to map high-dimensional inputs to outputs. The second reason flows from the first. Because of this mapping capability, practitioners often need to perform fewer manual steps in order to supply appropriate inputs, as compared to simpler models such as linear regressors. Although generally useful, this property is essential in situations where we have inputs for which we do not generally know how to consistently and efficiently extract relevant features from. Such input classes often are (nearly) unconsciously processed by what Kahneman (2011) would call 'system 1', such as the just-discussed visual and auditory stimuli. However, DNNs may be of use in any problem domain where we have few a priori insights in what may explain input-output relations. Think, for example, also of weather (H. Wang, Lei, Zhang, Zhou, & Peng, 2019) or stock market prediction (Chong, Han, & Park, 2017).

A subdomain of deep learning known as supervised deep learning has been most intensively studied in the last two decades (Smith, 2020), and within this subdomain most of the aforementioned successes have been obtained. However, some important challenges seem to persist as well (Adadi, 2021; Marcus, 2018).

One of the challenges is data-hungriness, which refers to the issue that often a significant amount of input-output pairs need to be supplied to DL models for training before they reach reasonable performance. Although we stated above that DL has been enabled, in part, by exposure of large, high-quality datasets, the problem persists because such datasets cannot be readily prepared for every problem domain. For example, collecting large amounts of data points in the healthcare domain is difficult in part due to data privacy requirements. In a sense, collecting enough samples is besides the point: intuitively, modern AI systems should be able to learn more efficiently, because we know animals are able to learn from just a couple of examples. As Yoshua Bengio phrases it in Smith ((2020), par. 2): "[...] Humans don't need that much supervision."

Besides data-hungriness, another challenge for especially supervised DL systems is explainability (Gilpin et al., 2018). In their paper, Gilpin and co-authors suggest to distinguish interpretability from explainability. They roughly define interpretability as "[...] the science of comprehending what a model did (or might have done)". Explainability instead requires "[...] models that are able to summarise the reasons for neural network behaviour, gain trust of users, or produce insights about the causes of their decisions" (Gilpin et al., 2018, both on p. 80). Explainability is a desirable and often crucial property of models, as such models may be easier to debug, more straightforward to understand by expert and non-expert users alike, and less prone to systemic biases, as such biases are exposed more readily (Adebayo et al., 2018).

Since the aforementioned two challenges appear not easily solvable under supervised deep learning, multiple prominent researchers have suggested to consider alternative, non-supervised deep learning methods (Smith, 2020). For example, Turing award-winner LeCun's "[...] money is on self-supervised learning, for machines to learn by observation, or learn without requiring so many labelled samples" (Smith, 2020, par. 7).

One alternative to supervised deep learning that may be particularly interesting is deep reinforcement learning (DRL). In order to understand DRL, one first needs to grasp what reinforcement learning (RL) is. One working definition may be "a field of machine learning [ML] that focuses on how to map [states] to actions [...] so as to maximise a numerical reward signal", paraphrasing (Sutton & Barto, 2018, p. 1). Deep RL, then, is the involvement of DNNs in this mapping, often to estimate how much reward actions in states tend to produce (Mnih et al., 2015). DRL is of interest in our search to alternatives to supervised DL, as it at least may address the issue of data-hungriness. This is thanks to the fact that, in (D)RL, we do not work with labeled input-output pairs but with reward signals that are generated by an environment. Crucially, the implementation of such reward signals may sometimes be more straightforward to obtain than the large amounts of samples required by supervised DL. For example, in robotic control, it may be more straightforward to signal a negative reward on contact with a rigid surface instead of annotating frames of a robotic arm reaching dangerously close to such a surface. In these cases, the DRL approach is less data-hungry in the sense that we annotated samples follow automatically once the reward function is specified. Here, of course, we rely on the reward function being easily characterised, which is not always true.

However, RL methods that incorporate DNNs directly still face the explainability challenge. Moreover, learning may take significant amounts of time—running programs for more than a month is not uncommon for the just-mentioned Atari environment, for example (Mnih et al., 2015, p. 534; Machado et al., 2018, p. 532).

An avenue of research within DRL that may address all three problems may be called 'object-based DRL'. With this approach, we exploit the fact that in deep reinforcement learning, the mapping from states to actions to take in those states may actually be broken down into two steps. First, we encode the states into a high-level representation, and then, in the second step, we use this encoding of the state to make decisions. This strategy intuitively appeals to how humans make decisions at the level of 'objects' instead of low-level, minute details. In playing video games, for example, a human player would probably not deem singular changes in pixel colors significant—it would instead reason about the playable character and how it interacts with opponents and items. Multiple authors have taken an 'object-based' approach to DRL and obtained promising results; such studies range from playing games to robotic control (Li, Sycara, & Iyer, 2017; Goel,

Weng, & Poupart, 2018; Lample & Chaplot, 2017; Guo, Dong, Chen, & Li, 2019).

Motivated by the results obtained in object-based DRL, we formulate two research questions based on the two challenges of DRL that we identified above. Particularly, these two questions are:

1. Does representing the state by its high-level objects accelerate learning in deep reinforcement learning methods, and
2. Can these high-level object representations make deep reinforcement learning methods more explainable?

To make the investigation into these two research questions concrete, we will work with and build on top of two relatively recent papers that utilise an 'object-based' approach, namely Li et al. (2017) and Goel et al. (2018). Furthermore, we use a set of baselines provided by Castro, Moitra, Gelada, Kumar, and Bellemare (2018)'s Dopamine framework to facilitate comparison to non-object-based approaches. Apart from this, we must stress that our aim is to provide an answer to the first research question, while we are only in the position to give a tentative answer to the second question.

In addition to testing both Li et al. (2017) and Goel et al. (2018)'s approach in the context of our two research questions, we seek to contribute to the research community by introducing a third approach—essentially a further-simplified version of (Li et al., 2017)'s architecture—which we involve in our experimentation as well. Moreover, we propose and implement three improvements for the computation of object saliency maps (OSMS)—the object-based explainability tool pioneered by Li and colleagues—and critically discuss their merit as such a tool.

This thesis, then, is structured as follows. We begin in the Theory by giving a brief exposition of some of the central principles that underpin reinforcement learning. The Related Work resumes from where the Theory ends by covering two papers that have influenced the trajectory of deep reinforcement learning (Mnih et al., 2015; Hessel et al., 2018), along with the work done by Li et al. (2017) and Goel et al. (2018). Then follows the Methdology, which explains in detail how we operationalise the two research questions posed above, building on the just-established Theory and Related Works. Subsequently, we present our findings in the Results, which is then succeeded by the Discussion. Finally, we close this thesis in the Conclusion.

Theory

Markov Decision Processes (MDPs)

In order to understand the problem reinforcement learning tries to solve, we need to understand Markov Decision Processes (MDPS; Bellman, 2010; Puterman, 1994). As the name implies, an MDP is a special type of stochastic process. Generally, stochastic processes are defined as ordered sets of random variables. In the case of MDPS, we have an ordered set of the form

$$\{ (R_t, S_t, A_t) \mid t \in \mathcal{T} \}, \tag{1}$$

where R_t, S_t, and A_t are the reward, state, and action at time step t. The values that the rewards, states, actions, and time steps can take on—respectively, $\mathcal{R}$, $\mathcal{S}_{\text{all}}$, $\mathcal{A}$, and $\mathcal{T}$—depend on the problem that is modelled using the MDP framework. For instance, $\mathcal{S}_{\text{all}}$ may be the set of points in $[0, 255]^{210\times160\times3}$—the possible images that can be displayed by a single Atari 2600 game frame; similarly, $\mathcal{T}$ is often the set of natural numbers, $\mathbb{N}$, representing discrete time.

Three notes need to be made at this point. First, we will use $\mathcal{T} = \mathbb{N}$ within this thesis. Second, although not generally necessary, we will always set R_0 to 0; it is a preset value and not part of the random variables. Third, we have made the set of possible actions $\mathcal{A}$ uniform across all possible states $\mathcal{S}_{\text{all}}$. This need not be true in more rigorous treatments of MDPS: one can imagine that only certain strict subsets of $\mathcal{A}$ are legal within certain strict subsets of $\mathcal{S}_{\text{all}}$. In that case, we could instead write $\mathcal{A}(\mathbf{s}) \subseteq \mathcal{A}$, for all $\mathbf{s} \in \mathcal{S}_{\text{all}}$, to make this point explicit. Within this thesis, this uniformity assumption is appropriate and thus used.

The connection between successive triples in an MDP is defined by means of a so-called transition function,

$$p_t(\mathbf{s}', r \mid \mathbf{s}, a) \stackrel{\text{def}}{=} \mathrm{P}(S_t = \mathbf{s}', R_t = r \mid S_{t-1} = \mathbf{s}, A_{t-1} = a), \tag{2}$$

for all $\mathbf{s}, \mathbf{s}' \in \mathcal{S}_{\text{all}}$, $r \in \mathcal{R}$, $a \in \mathcal{A}$, and $t \in \mathcal{T} \setminus \{0\}$. Here, we notationally view states as vectors and actions and rewards as scalars. This is of course not generally applicable, but it is in the context we will apply the MDP framework to. In words, p maps state-action-reward-next state quadruples to values within $[0, 1]$, effectively assigning a probability of transitioning from $(\mathbf{s}, a)$ to $\mathbf{s}'$, obtaining reward r along the way. Notice that this transition function definition shows why the process is regarded as Markovian: we rely only on the directly preceding state of $\mathbf{s}'$, namely $\mathbf{s}$; no states preceding $\mathbf{s}$ influence $p_t(\mathbf{s}', r \mid \mathbf{s}, a)$. Note further the subscript on p_t: in the general case, the transition function may adapt as t increases. If this is not the case, we may drop the t subscript.

By now, it may be clear what the terms 'Markovian' and 'process' pertain to within the MDP framework. Equally important in MDPS, however, is the decision maker. Given the aforementioned stochastic, Markovian process, the objective of the decision maker is to select actions within visited states in such a way that the cumulative future-discounted reward is maximised, in a manner to be made precise later. This aggregation of rewards, also known as the return, is defined as

$$G_t \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{3}$$

where $0 \leq \gamma < 1$ is a hyperparameter known as the discount rate, controlling how near- or far-sighted the decision maker is: the lower γ is, the more the decision maker values near-immediate rewards over longer-term rewards. An MDP with $\gamma = 0$ is known as an undiscounted Markov Decision Process, whereas one with $\gamma \neq 0$ is a discounted one; most MDPs are of the latter type. Besides this, t is any value in $\mathcal{T}$.

The decision maker may influence its trajectory through the MDP—and therefore its returns—by selecting actions according to a decision function $\pi(a \mid \mathbf{s})$. This is a function that maps states to actions to take in those states; it may either be probabilistic or deterministic. If both $\mathcal{S}_{\text{all}}$ and $\mathcal{A}$ store discrete values, then π is a probability mass function (PMF); otherwise it is a probability density function (PDF) in the states, actions, or both. Also, if we fix π, the MDP reduces to a Markov chain, with only the transition function p_t affecting possible outcomes in successor triples $(R_{t+1}, S_{t+1}, A_{t+1})$.

Before moving on, we must note that MDPs come in two variants: episodic and continuing. In the former type, a strict subset of the states $\mathcal{S}_{\text{all}}$ is known as the set of terminal states. If the decision maker visits such a state, the sequence of state-action-reward triplets stops. This does not occur in the continuing type of MDPs, where sequences proceed indefinitely. From now on, if we consider episodic MDPs, let

$$\begin{aligned} &[0, T_0], [T_0 + 1, T_1], \ldots, [T_{n-1} + 1, T_n], \ldots \\ &\text{with } T_{-1} \stackrel{\text{def}}{=} -1,\ T_0, T_1, \ldots \in \mathcal{T},\ \text{and } n \in \mathbb{N} \end{aligned}$$

represent the episodes' time step ranges, with $T_0, T_1, \ldots$ representing time steps at which the decision maker encounters terminal states. The symbol T_{-1} is notationally introduced to allow us to write time step intervals covering episodes; it does not represent a true time step in $\mathcal{T}$. At terminal states $\mathbf{s}_{T_0}, \mathbf{s}_{T_1}, \ldots$ any action may be taken (but is not 'used' further in the process), and the reward is always set to zero: $R_{T_n} = 0$, for all $n \geq 0$. Then, at time steps $T_0 + 1, T_1 + 1, \ldots \in \mathcal{T}$, the decision maker starts with a new instance of the stochastic process, although it itself and its decision function carry over from the just-ended episode. Notice that, possibly against one's intuition, time does not reset per episode. This is purely done to enable certain mathematical expressions. Furthermore, we must provide an alternative definition of return for the episodic case. Particularly, if we use G_t with $t \in [T_{n-1} + 1, T_n]$, for any $n \geq 0$, then the episodic return's definition is

$$G_t \stackrel{\text{def}}{=} \begin{cases} \sum_{k=t+1}^{T_n} \gamma^{k-t-1} R_k & \text{if } t \in [T_{n-1} + 1, T_n\rangle, \\ 0 & \text{if } t = T_n, \end{cases} \tag{4}$$

with the same qualification on γ as in Equation 3. We require one last notational addition to include episodic MDPs in our treatment. From now on, let $\mathcal{S}$ encompass all non-terminal states and let $\mathcal{S}_{\text{terminal}}$ denote all terminal states. Note that $\mathcal{S}_{\text{all}} = \mathcal{S} \cup \mathcal{S}_{\text{terminal}}$. For continuing tasks, $\mathcal{S}_{\text{terminal}} = \emptyset$.

A natural observation to make is that not all decision functions are equal in the returns they are probable to generate. In order to determine which decision functions are relatively 'good'—in the sense that their returns are probable to be relatively large—we introduce two important functions. First, the value function (of a state $\mathbf{s}$, with respect to a policy π):

$$\begin{aligned} v_\pi(\mathbf{s}) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = \mathbf{s}], \\ &\text{for all } \mathbf{s} \in \mathcal{S},\ t \in \mathcal{T},\ \text{and } \pi \in \mathcal{P}, \end{aligned} \tag{5}$$

where $\mathcal{P}$ is a new symbol representing the set of all possible policies. $v_\pi(\mathbf{s})$ expresses the expected return of making decisions using decision function π after having visited state $\mathbf{s}$. Intuitively, $v_\pi(\mathbf{s})$ simultaneously expresses how 'potent' state $\mathbf{s}$ is in terms of returns to be gained from it, as well as how 'effective' decision function π is in leveraging states to obtain returns. Note that these two views are strongly related: a state from which only low returns can be extracted make any decision function seem ineffective, and, vice-versa, a policy that cannot extract high returns in any state may cause all states to have a relatively low v_π-valuation. Apart from this, note that the valuation of any terminal state $\mathbf{s} \in \mathcal{S}_{\text{terminal}}$ is zero, because no further rewards can be gained once such a state is visited.

The second π-evaluating function is very similar to $v_\pi(\mathbf{s})$. It is known as the action-value function (of a state-action pair $(\mathbf{s}, a)$, with respect to a policy π) and it defined as follows:

$$\begin{aligned} q_\pi(\mathbf{s}, a) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = \mathbf{s}, A_t = a], \\ &\text{for all } \mathbf{s} \in \mathcal{S},\ a \in \mathcal{A}, t \in \mathcal{T},\ \text{and } \pi \in \mathcal{P}. \end{aligned} \tag{6}$$

Akin to $v_\pi(\mathbf{s})$, $q_\pi(\mathbf{s}, a)$ expresses the expected return of taking action a upon visiting state $\mathbf{s}$, and following decision function π from that point onward. Further, as is the case with $v_\pi(\mathbf{s})$, $q_\pi(\mathbf{s}, a) = 0$ when $\mathbf{s} \in \mathcal{S}_{\text{terminal}}$, for all $a \in \mathcal{A}$.

An important observation pertaining to $v_\pi(\mathbf{s})$ and $q_\pi(\mathbf{s}, a)$ is that, because of their definitions, they can be recursively expanded (Bellman, 2010; Sutton & Barto, 2018, pp. 59 ff.). To begin, for $v_\pi(\mathbf{s})$ we have

$$\begin{aligned} v_\pi(\mathbf{s}) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = \mathbf{s}] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = \mathbf{s}] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid \mathbf{s}) \sum_{\mathbf{s}' \in \mathcal{S}, r \in \mathcal{R}} p(\mathbf{s}', r \mid \mathbf{s}, a) \\ &\qquad \cdot (r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = \mathbf{s}']) \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid \mathbf{s}) \sum_{\mathbf{s}' \in \mathcal{S}, r \in \mathcal{R}} p(\mathbf{s}', r \mid \mathbf{s}, a)\,(r + \gamma v_\pi(\mathbf{s}')), \\ &\text{for all } \mathbf{s} \in \mathcal{S},\ t \in \mathcal{T},\ \text{and } \pi \in \mathcal{P}, \end{aligned} \tag{7}$$

where we applied the definition of $v_\pi(\mathbf{s})$—Equation 5—in the transition from the one-to-last line to the last line, noting that

$\mathbf{s}'$ is the state at $t+1$. Similarly, for $q_\pi(\mathbf{s}, a)$ we obtain

$$\begin{aligned} q_\pi(\mathbf{s}, a) &\stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t \mid S_t = \mathbf{s}, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = \mathbf{s}, A_t = a] \\ &= \sum_{\mathbf{s}' \in \mathcal{S}, r \in \mathcal{R}} p(\mathbf{s}', r \mid \mathbf{s}, a) \sum_{a' \in \mathcal{A}} \pi(a' \mid \mathbf{s}') \\ &\qquad \cdot (r + \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = \mathbf{s}', A_{t+1} = a']) \\ &= \sum_{\mathbf{s}' \in \mathcal{S}, r \in \mathcal{R}} p(\mathbf{s}', r \mid \mathbf{s}, a) \sum_{a' \in \mathcal{A}} \pi(a' \mid \mathbf{s}') \left(r + q_\pi(\mathbf{s}', a')\right), \end{aligned} \tag{8}$$

$$\text{for all } \mathbf{s} \in \mathcal{S},\ a \in \mathcal{A},\ t \in \mathcal{T},\ \text{and } \pi \in \mathcal{P}.$$

Again, we re-invoke the definition of the (action-)value function—Equation 6 this time around—to obtain an inward-expanded variant of $q_\pi(\mathbf{s}, a)$. Together, Equations 7 and 8 demonstrate the Bellman equation for the state- and action-value functions, respectively. It will be used later to motivate certain optimisation strategies for MDPs.

We are now in the position to express more precisely when we regard a decision function as being optimal. Let $\pi, \pi' \in \mathcal{P}$ be two policies. Then define the binary relation $\geq : \mathcal{P} \times \mathcal{P} \to \{\top, \bot\}$, where $\pi \geq \pi'$ is true if and only if $v_\pi(\mathbf{s}) \geq v_{\pi'}(\mathbf{s})$, for all $\mathbf{s} \in \mathcal{S}$. Then, if for some policy $\pi^* \in \mathcal{P}$ it is true that

$$\begin{aligned} &v_{\pi^*}(\mathbf{s}) \geq v_{\pi'}(\mathbf{s}) \text{ for all } \mathbf{s} \in \mathcal{S} \text{ and all } \pi' \in \mathcal{P} \\ &\Leftrightarrow \\ &\pi^* \geq \pi', \text{ for all } \pi' \in \mathcal{P}, \end{aligned} \tag{9}$$

then we say that the policy π^* is optimal with respect to the MDP it is trying to maximise expected return for.

It is important to note that multiple optimal decision functions may exist with respect to the MDP under consideration. The value function for such optimal policies is known as the optimal state-value function, or $v_*(\mathbf{s})$. It is additionally known that all such optimal policies have an identical, so-called optimal action value function, $q_*(\mathbf{s}, a)$, as well (Sutton & Barto, 2018, pp. 62–63).

Given the notion of an optimal policy $\pi^* \in \mathcal{P}$, the problem posed by MDPs is this. Start at time $t = 0$. Let the decision maker sequentially receive triples $(r_t, \mathbf{s}_t, a_t)$ via $p_t(\mathbf{s}_t, r_t \mid \mathbf{s}_{t-1}, a_{t-1})$, $\pi_{t-1}(a_{t-1} \mid \mathbf{s}_{t-1})$ and $\pi_t(a_t \mid \mathbf{s}_t)$, where the decision maker only sees the triples, and often has no knowledge of p_t. Design an algorithm by which the decision maker updates its π_{t-1} to π_t given triples generated up until t by the MDP, so as to obtain—or approximate as closely as possible—one of the π^* as defined by Equation 9. Note that, if the problem is episodic, it continues across episodes: the triples are reset, but the decision maker and decision function remain in the configurations they were in at the end of the terminated episode.

A straightforward and exact solution to the MDP problem is to solve the so-called Bellman optimality equations via dynamic programming; these are special cases of the Bellman equations for when $v_\pi = v_{\pi^*}$ and $q_\pi = q_{\pi^*}$, for some optimal policy $\pi^* \in \mathcal{P}$ (Sutton & Barto, 2018, pp. 63–67). However, this approach requires perfect knowledge of the transition function p_t—something that is often not applicable. Even if p_t were available, solving the problem exactly via dynamic programming can become computationally unwieldy beyond toy problems. Thus, we need more sophisticated approaches.[1] Before investigating more realistically applicable solutions, we relate the MDP problem to the reinforcement learning problem, and continue with the latter notion within this thesis.

The reinforcement learning problem

The preceding discussion on MDPs provides a minimal, mathematical fundament necessary to understand the reinforcement learning problem. However, to transition from the general mathematical topic of MDPs to reinforcement learning, we need to shift in perspective and terminology: although the mathematics remain very similar, conceptually, the two fields are quite distinct. We now briefly detail this conceptual difference.

The challenge of finding (near-)optimal decision functions, in the sense of Equation 9, can be regarded as a pure, mathematical task of finding a $\pi \in \mathcal{P}$ that seeks to maximise the expectation over future-discounted reward random variables $R_0, R_1, \ldots$, considering the stochasticity of the process, embodied in the transition function p_t. This search is carried out by an algorithm that we previously called the decision maker.

In contrast, the (D)RL field views the decision maker first and foremost as an active, autonomous system that adapts its behaviour in response to feedback; the agent is implemented as an algorithm, but it is viewed as an entity.

Concretely, this changes the terminology in the following way. Previously, we said that a decision maker transitioned from some reward, state, action triple $(r, \mathbf{s}, a) \in \mathcal{R} \times \mathcal{S} \times \mathcal{A}$ to another successor triple $(r', \mathbf{s}', a') \in \mathcal{R} \times \mathcal{S}_{\text{all}} \times \mathcal{A}$[2] due to the transition function $p(\mathbf{s}', r' \mid \mathbf{s}, a)$ and the decision function $\pi(a \mid \mathbf{s})$. In the terminology of (D)RL, we instead say that an agent took action a in state $\mathbf{s}$ by following its policy $\pi(a \mid \mathbf{s})$. In turn, it received reward r' and moved on to state $\mathbf{s}'$ according to the environment's dynamics, dictated by $p(\mathbf{s}', r' \mid \mathbf{s}, a)$.

[1] We remark that a sub-branch of dynamic programming, known as approximate dynamic programming, may be used if one wishes to continue using dynamic programming when time and memory complexity increase. See, for an exposition, Buşoniu, Babuška, de Schutter, and Ernst (2010).

[2] The successor state may be a terminal state, while the preceding state may not. That is, unless we say that terminal states deterministically transition to themselves indefinitely and with zero reward, as discussed above in the unification of episodic and continuous problems.

Given this shift in perspective, we can rephrase the MDP problem. Start at time $t = 0$. Let the agent continually observe the environment, producing, at time t, state $\mathbf{s}_t$. Based on this state, the agent takes action a_t, drawn from its policy $\pi_t(a_t \mid \mathbf{s}_t)$, obtaining reward r_{t+1} and observing next state $\mathbf{s}_{t+1}$ in doing so. Design an agent which adapts its policy from π_t to π_{t+1} such that the policy converges to—or approximates as closely as possible—one of the π^* as defined by Equation 9. As was the case in the original MDP problem, the agent and policy 'carry over' from episode ends to next episodes if the problem is episodic.

The just-formulated task is known as the reinforcement learning problem. As was mentioned in the MDP section, it is often impractical to find a policy that exactly equals one of the π^*, primarily due to memory and computational constraints. For instance, if we would keep track of our approximations of $q_\pi(\mathbf{s}, a)$, for every $(\mathbf{s}, a)$-pair, the time and memory demands of our program would grow at least proportional to $|\mathcal{S}| \times |\mathcal{A}|$, which can be significant if either $\mathcal{S}$ or $\mathcal{A}$ (or both) become large sets. Thus, we often seek to approximate the π^*s as close as possible. This 'weaker' task formulation is also adopted within this thesis.

Two solution classes

As mentioned above, dynamic programming is a solution that, in principle, is capable of solving the reinforcement problem. It assumptions often do not hold, though—perhaps first and foremost the requirement that the environment's p_t is given in full. Within this subsection, we present two reinforcement learning solution classes that do not require this model to be present. They are known as Monte-Carlo methods and temporal difference methods. Although they both learn environmental dynamics from experience instead of from specification, they can be viewed as polar opposites of one another.

Monte-Carlo methods. Perhaps the primary characteristic of agents from the Monte-Carlo solution class is that they adapt their policies only after episodes have ended. Note that as a direct consequence, this approach to reinforcement learning is not well-defined for continuing problems. Further, we could view Monte-Carlo methods as sampling full returns from all possible $(\mathbf{s}, a) \in \mathcal{S} \times \mathcal{A}$ pairs defined for the problem that is being addressed.

In order to explain Monte-Carlo methods formally, we begin by noting that, within Monte-Carlo methods, policies are fixed within single episodes. As such, let us introduce the notation $\pi_n \in \mathcal{P}$ to denote the policy used within n^{th} episode, with $n \geq 0$. Then realise that policy π_n is in use at time steps in the range $[T_{n-1} + 1, T_n]$. Given this, we follow the division of concerns used in Sutton and Barto (2018). Specifically, we first consider the problem of making an estimate V_n of v_{π_n}, and, analogously, of making an estimate Q_n of q_{π_n}. We subsequently find a way by which we can update π_n to π_{n+1}, in order for it to be used in the next episode. Finally, we combine the previous two steps to arrive at a complete control algorithm to which a Monte-Carlo method can adhere.

We start with the estimating v_{π_n} using V_n. If the agent only collects full episodic returns $g_0, g_{T_0+1}, \ldots$, then there is an intuitive way to estimate v_{π_n}. Begin by observing that, at the end of episode n, returns $g_0, \ldots, g_{T_{n-1}+1}$ are known. Given these, we can, per state $\mathbf{s} \in \mathcal{S}$, collect returns starting from $\mathbf{s}$ per each episode. We do so by finding per $g \in g_0, \ldots, g_{T_{n-1}+1}$ the first time step after having visited $\mathbf{s}$—call this step $t_{g,\mathbf{s}} \in \mathcal{T}$, and taking the arithmetic mean of the returns from these time steps onward:

$$\begin{aligned} V_n(\mathbf{s}) &\stackrel{\text{def}}{=} \frac{\sum_{g \in \{g_0, \ldots, g_{T_{n-1}+1}\}} g_{t_{g,\mathbf{s}}}}{n+1} \\ &\approx \mathbb{E}_{\pi_n}[G_t \mid S_t = \mathbf{s}] \\ &\approx v_{\pi_n}(\mathbf{s}), \\ &\text{for all } \mathbf{s} \in \mathcal{S},\ t \in \mathcal{T},\ \text{and } n \geq 0. \end{aligned} \tag{10}$$

It may happen that state $\mathbf{s}$ is never visited in some return g. In that case, we let $g_{t_{g,\mathbf{s}}} = 0$. The approximation given in Equation 10 is known to converge to v_{π_n} as the collection of first visits to states $\mathbf{s}$ tends to infinity (Sutton & Barto, 2018, p. 93). Another similar approximation exists that uses all visits to $\mathbf{s}$ within episodes—not just the first one—but we omit that case here.

Our motivation for using state- and action-value functions is to inform updates to policy. In any method that does not assume a model—including Monte-Carlo methods—we as such cannot use V_n, because it stores no information on the value of actions performed in states. Thus, we use the closely related estimation Q_n instead:

$$\begin{aligned} Q_n(\mathbf{s}, a) &\stackrel{\text{def}}{=} \frac{\sum_{g \in \{g_0, \ldots, g_{T_{n-1}+1}, g_{t_{g,\mathbf{s},a}}\}} g_{t_{g,\mathbf{s},a}}}{n+1} \\ &\approx \mathbb{E}_{\pi_n}[G_t \mid S_t = \mathbf{s}, A_t = a] \\ &\approx q_{\pi_n}(\mathbf{s}, a), \\ &\text{for all } \mathbf{s} \in \mathcal{S},\ a \in \mathcal{A},\ t \in \mathcal{T},\ n \geq 0, \end{aligned} \tag{11}$$

where we use the symbol $g_{t_{g,\mathbf{s},a}}$, with $t_{g,\mathbf{s},a} \in \mathcal{T}$, to denote the cumulative, future-discounted return starting from state-action pair $\mathbf{s}, a \in \mathcal{S} \times \mathcal{A}$ in the trajectory that yielded return g. $Q_n(\mathbf{s}, a)$ has a similar convergence guarantee as $V_n(\mathbf{s})$. We may now use $Q_n(\mathbf{s}, a)$ to update π_n to π_{n+1}. The naive approach would be to, per state $\mathbf{s} \in \mathcal{S}$, select that action $a \in \mathcal{A}$ that produces the maximal $Q_n(\mathbf{s}, a)$—a fully greedy policy update. This course of action is problematic however, because it produces policies that never select certain state-action pairs. If the optimal policy includes such never-selected pairs, then greedy policy updating gets stuck in a 'local optimum' of policy-space. Instead, we must ensure a minimal level of exploration. Two ways of doing so are: (1) Ensuring a nonzero probability of the agent starting in state-action pair $(\mathbf{s}, a) \in \mathcal{S} \times \mathcal{A}$, for all possible pairs. This can

be an unrealistic requirement, especially if the state-action pair-space is considerably large. (II) Guaranteeing a minimal level of action-exploration per state. This can be done by making π_n ε-soft, which would impose the requirement that

$$\pi_n(a \mid \mathbf{s}) \geq \frac{\varepsilon}{|\mathcal{A}|}, \quad \text{for all } \mathbf{s} \in \mathcal{S} \text{ and } a \in \mathcal{A}.\ \varepsilon \in \langle 0, 1\rangle. \tag{12}$$

The ε-greedy method of updating to π_{n+1} is one way to compromise between exploration and exploitation:

$$\pi_{n+1}(a \mid \mathbf{s}) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q_n(\mathbf{s}, a'), \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{otherwise.} \end{cases} \tag{13}$$

We present this ε-greedy updating method as an exemplary approach to updating the policy; other procedures exist as well.

Given our method of obtaining $Q_n(\mathbf{s}, a)$, as well as an approach to updating π_n to π_{n+1}, we can simply cycle between policy evaluation (producing Q_n) and policy improvement (yielding π_{n+1}), for all $n \geq 0$. The specific algorithm may differ depending on how, for example, exploration is guaranteed, but this is a brief and minimal exposition of the primary mechanism behind a Monte-Carlo agent.

Temporal-difference learning methods. As mentioned before, Monte-Carlo methods have a great advantage over analytical, dynamic programming methods: they learn the environmental dynamics instead of needing these to be specified. Despite this, the Monte-Carlo solution class has a weakness that may become problematic in certain problem domains: they work on full, sampled returns instead of individual rewards. Besides that it would be convenient to update more frequently, it may also result in more efficient learning, because a relatively rapidly updated policy may take on more 'informative' trajectories.

An extreme example: imagine a policy that has been initialised unfortunately and that, as a result, remains stuck in a heavily penalising state throughout the complete first episode. Monte-Carlo methods would need to await the whole episode before being able to rectify behaviour. If we instead used a faster-updating method, we could correct the policy already during the episode. This would possibly cause the agent to experience other states beside the penalising state; those other states could then also be updated during the first episode while the Monte-Carlo method would never have observed them.

This intuition motivates the solution class of temporal-difference (TD) learning methods. Because TD learning methods update more frequently than after every episode, instead of using the symbols Q_n, $q_{\pi_n}(\mathbf{s}, a)$, and $\pi_n(a \mid \mathbf{s})$, we instead use Q_{t+m}, $q_{\pi_{t+m}}(\mathbf{s}, a)$, and π_{t+m}, with $t \in \mathcal{T}$ and $m \geq 1$. m represents the number of steps we consider the update to π_t over. As was the case with the Monte-Carlo solution class, we first discuss the evaluation problem, move on to the policy improvement problem, and finally arrive at a specification for agent control by combining the two steps.

We begin, again, with the action-value estimation problem. In the Monte-Carlo case, we used the arithmetic mean of first-visits of state-action pairs across sampled trajectories to obtain estimates $Q_n(\mathbf{s}, a) \approx q_{\pi_n}(\mathbf{s}, a)$. Within TD learning, we instead estimate the action-value function for $t + m$ time steps from now. In order to do so, we leverage the definition of return (Equation 3) to introduce the symbol $G_{t:(t+m)}$. In words, it expresses a future-discounted reward accumulation starting from step t and adding up until $t + m - 1$, after which the remaining rewards may be replaced by our Q_{t+m-1}-estimation of the remainder of the return, G_{t+m-1}. Mathematically, the definition of $G_{t:(t+m)}$ is

$$\begin{aligned} G_{t:(t+m)} \overset{\text{def}}{=}\ & R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{m-1} R_{t+m} + \\ & \gamma^m \mathbb{E}_\pi[G_{t+m-1} \mid S_{t+m} = \mathbf{s}_{t+m},\ A_{t+m} = a_{t+m}], \\ & \text{for all } t \in [0,\ T_0 - m\rangle \cup [T_0 + 1,\ T_1 - m\rangle \cup \cdots \\ & \text{and } m \geq 1. \end{aligned} \tag{14}$$

In the special case of $G_{t:(t+m)}$ with $t \not< T_n - m$, for any $m \geq 0$, $G_{t:(t+m)}$ is defined to simply be G_t; in estimating $G_{t:(t+m)}$, we would drop the G_{t+m-1} term in that case. Given this symbol, we can finally specify how to estimate $q_{\pi_{t+m}}(\mathbf{s}, a)$. We do so by defining $Q_{t+m}(\mathbf{s}_t, a_t)$ as

$$\begin{aligned} Q_{t+m}(\mathbf{s}_t, a_t) \overset{\text{def}}{=}\ & Q_{t+m-1}(\mathbf{s}_t, a_t) \\ & + \alpha\Big(r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^m Q_{t+m-1}(\mathbf{s}_{t+m}, a_{t+m}) \\ & - Q_{t+m-1}(\mathbf{s}_t, a_t)\Big), \end{aligned} \tag{15}$$

with the same qualifications for t and m as were the case in Equation 14, and where $\alpha \in [0, 1]$ is called the learning rate. Commonly, $\alpha \ll 1$.

Equation 15 shows why the solution class of temporal-difference learning methods is named the way it is: the approach to estimating the action-value function is incremental, with successive corrections being made to action-value estimates $Q(\mathbf{s}, a)$. These corrections are made by considering the discrepancy (thus, 'temporal difference') between (I) the accumulated, future-discounted rewards of the last m time steps plus a future-discounted stand-in for G_{t+m}, namely the estimate $Q_{t+m-1}(\mathbf{s}_{t+m}, a_{t+m})$, and (II) our current estimate $Q_{t+m-1}(\mathbf{s}_t, a_t)$. The former is sometimes referred to as the target value; the latter is known as the predicted or estimated value. Their difference is used to adjust Q_{t+m} over the time steps within an episode. Notice that the agent updates its action-value estimates starting from time step m, and that the last $m - 1$ updates only get applied after an episode has ended, provided that the problem is episodic.

As was the case in the Monte-Carlo solution class, the policy improvement step builds upon the result of the action-

value estimation step. We can once again choose an ε-soft policy, with $\varepsilon \in \langle 0, 1\rangle$. Alternatively, we can separate the policy π_{t+m} into two: a target policy $\pi^{\text{target}}_{t+m}$ and a behaviour policy $\pi^{\text{behaviour}}_{t+m}$. We can optimise the former greedily with respect to Q_{t+m-1}, while the latter remains ε-soft. However, doing so requires taking into account the probabilistic difference in action selections per state between the two policies.[3] Apart from this, TD control follows from cycling back and forth between action-value estimation and acting on these estimations using a scheme such as ε-soft.

A special case of temporal-difference learning is known as Q-learning (Watkins, 1989). The method is historically significant due to it being the first off-policy temporal-difference learning method. It also does not use importance sampling, as suggested above. Let us consider the $m = 1$ case of Q-learning. In this method, Q_{t+1}-estimates are made according to

$$Q_{t+1}(\mathbf{s}_t, a_t) = Q_t(\mathbf{s}_t, a_t) + \alpha\Big(r_{t+1} + \gamma \max_{a\in\mathcal{A}} Q_t(\mathbf{s}_{t+1}, a) - Q_t(\mathbf{s}_t, a_t)\Big), \tag{16}$$

where one should note the maximisation over the action-value estimation; in the general $m \geq 1$ action-value estimation method presented earlier, we did not use such a maximum. This maximisation is precisely the reason why Q-learning may be used off-policy: the maximum operator implicitly makes policy improvement strictly greedy with respect to the Q-values. Crucially, however, we can introduce another behaviour policy that determines the action to take next; it need not be equal to the implicit action-value-updating policy.

Related Work

Deep Q-networks

Most of the papers discussed in these Related Works extend the work done by Mnih and colleagues on deep Q-networks (DQNS) (Mnih et al., 2013, 2015). As such, we start by considering this seminal work.

Motivated by advances in the visual and auditory problem domains, such as in Krizhevsky et al. (2012), the authors pose the question of how to extend deep learning to the domain of reinforcement learning. The challenges identified thereafter flow directly from the fact that reinforcement learning is, in some regards, very different from regular supervised learning problems. First, reinforcement learning has no notion of 'output labels'; instead, agents receive rewards per each action taken. Although seemingly related, they function very differently from the perspective of model training: labels are always present (at least in the classic supervised learning setting), they are assumed to be the ground truth, and you can directly associate labels to inputs. Environmental rewards, in contrast, may only be dispensed sporadically, may be contradictory from one observation to the next, and may only be presented after the 'root cause action' has been taken many time steps back. Second, the input observations are highly correlated: having seen some $\mathbf{s}_t$, one can generally infer most of the content of $\mathbf{s}_{t+1}$. Further, those same observations may alter in character over time as the policy changes. For example, random policies may not get far into a video game environment, whereas a relatively well-performing policy encounters new levels with entirely different observations to be drawn from them.

The authors propose to combine two existing ideas to address these issues: Deep off-policy Q-learning, as introduced above, and experience replay (Lin, 1992; Riedmiller, 2005). We will now discuss these ideas in turn.

Deep off-policy Q-learning. Q-learning as presented earlier has an important flaw: as the state- or action-space (or both) get large, learning becomes a daunting task because all $(\mathbf{s}, a) \in \mathcal{S} \times \mathcal{A}$ action-values need to be learnt separately. Even with an exploring behaviour policy, the absence of generalisation across the pairs greatly inhibits learning. Since many interesting problem domains are characterised by large state-action spaces—think, for instance, of color images as observations—it would be desirable to somehow learn action-value functions without using straightforward but inefficient $(\mathbf{s}, a)$-lookups.

Let us concentrate for the moment on the implementation of the action-value function; we drop the time step subscript and re-introduce it later. One solution to replace the 'lookup version' of $Q(\mathbf{s}, a)$ is to use linear methods. For example, we could introduce a feature-extracting function $\text{F} : \mathcal{S} \to \mathbb{R}^F$. Given any state $\mathbf{s} \in \mathcal{S}$, we compute $F \geq 1$ features of $\mathbf{s}$ that somehow help in the estimation of the action-value function. We then can introduce a weights matrix $W \in \mathbb{R}^{|\mathcal{A}|\times F}$ that maps feature vectors to $|\mathcal{A}|$ Q-values, one for each action. In total, we then may implement $Q(\mathbf{s}, a; W)$, now parameterised on the features-to-action values weight matrix W, using a linear model

$$Q(\text{F}(\mathbf{s}), a; W) = (\, W\, \text{F}(\mathbf{s})\,)_a\,, \tag{17}$$

where the a-subscript means to take the entry for action a of the action-values vector. Then, instead of revising $Q(\text{F}(\mathbf{s}), a; W)$ 'classically' as in Equation 16, we can use stochastic gradient descent instead, assuming we try to minimise the squared error between the target $r_{t+1} + \gamma \max_{a\in\mathcal{A}} Q(\text{F}(\mathbf{s}_{t+1}), a; W_t)$ and our action-value estimate

[3] Technically, the use of an ε-soft policy is not strictly required. Instead, we must ensure that, for all states $\mathbf{s} \in \mathcal{S}$, whatever action is chosen in the here-assumed deterministic target policy $\pi^{\text{target}}_{t+m}$, $\pi^{\text{behaviour}}_{t+m}$ also may select action with nonzero probability. This is known as the principle of coverage.

$Q(\text{F}(\mathbf{s}_t), a; W_t)$:

$$\begin{aligned} W_{t+1} &\stackrel{\text{def}}{=} W_t - \alpha \frac{1}{2} \nabla_{W_t} \cdot \\ &\qquad \Big(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(\text{F}(\mathbf{s}_{t+1}), a; W_t) - Q(\text{F}(\mathbf{s}_t), a_t; W_t) \Big)^2 \\ &\neq W_t + \alpha \cdot \\ &\qquad \Big(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(\text{F}(\mathbf{s}_{t+1}), a; W_t) - Q(\text{F}(\mathbf{s}_t), a_t; W_t) \Big) \cdot \\ &\qquad\qquad \nabla_{W_t} Q(\text{F}(\mathbf{s}_t), a_t; W_t) \end{aligned} \tag{18}$$

Here, we notice a problem: we must also take the gradient of the maximum term $\max_{a \in \mathcal{A}} Q(\mathbf{s}_{t+1}, a; W_t)$ as it also is parameterised by W_t, but it is well known that taking (partial) derivatives and gradients of piecewise functions may lead to discontinuities in the resultant function. A workaround to this problem is to introduce another pair of weights $\overline{W}_t \in \mathbb{R}^{|\mathcal{A}| \times F}$ called the target weights, and replacing the maximisation's W_t by $\overline{W}_t$ instead. Once every pre-defined duration, the weights from the regular weights set W_t get copied to $\overline{W}_t$, effectively synchronising the two. This alternative update can then be used, avoiding this source of training instability.

Linear methods, however, are limited in two ways. First, we must somehow obtain an effective feature extraction function F. This often requires manual labour and is domain-specific. Further, as we know from the field of neural networks, linear methods are limited in effectiveness compared to non-linear models.

Hence, Mnih et al. propose to use a deep learning model, similar in spirit to the just-introduced linear model. Different from the linear model is that a deep convolutional neural network is used instead of a linear weights matrix. Let θ_t and $\overline{\theta}_t$ be the parameterisation of the online and target versions of this network at time step $t \in \mathcal{T}$. Then the update made for this network at t is

$$\begin{aligned} \theta_{t+1} &= \theta_t + \alpha \cdot \\ &\qquad \Big(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q\Big(\text{F}(\mathbf{s}_{t+1}), a; \overline{\theta}_t\Big) - Q(\text{F}(\mathbf{s}_t), a_t; \theta_t) \Big) \cdot \\ &\qquad\qquad \nabla_{\theta_t} Q(\text{F}(\mathbf{s}_t), a_t; \theta_t). \end{aligned} \tag{19}$$

For the sake of brevity, we abstain from discussing the network architecture, but we do present it diagrammatically in Figure 1, also to ease comparison with other, related networks. Full details can be found in Mnih et al. (2015).

Finally, we remark that this network is off-policy by the same reasoning we used for regular Q-learning above. This is relevant for the following idea: the use of experience replay.

Experience replay. Earlier we stated that successively visited states highly correlate in their content. This problem may be resolved if the quadruples $(\text{F}(\mathbf{s}_t), a_t, r_{t+1}, \text{F}(\mathbf{s}_{t+1}))$ supplied for training are not those that appear during online execution, but instead randomly-drawn earlier quadruples from all earlier episodes. This idea is known as experience replay (Lin, 1992; Riedmiller, 2005). In that way, more diverse situations are encountered during training, which makes the network less likely to overfit as it has considerably less opportunity to parameter-tune to successive entries in the trajectory. In order to allow this, however, we need to be able to decouple the agent's action-evaluations from its action-selections; in other words, we require an off-policy method. As we mentioned just now, DQN is indeed off-policy. In particular, the Q-learning algorithm underlying it has been chosen precisely because it allows for off-policy agent control.

Let $\mathcal{B} \in (\mathbb{R}^F \times \mathcal{A} \times \mathcal{R} \times \mathbb{R}^F)^{B_{\text{replay}}}$ be an experience replay buffer of size $B_{\text{replay}} \geq 1$. Ideally, $\mathcal{B}$'s size is unbounded, but computer memory constraints force us to use a large but finite value for B_{replay} instead. At every new time step $t \in \mathcal{T}$ within an episode we append the quadruple $(\text{F}(\mathbf{s}_{t-1}), a_{t-1}, r_t, \text{F}(\mathbf{s}_t))$ to the end of $\mathcal{B}$. Once the buffer's storage size B_{replay} is reached the oldest quadruples can be overwritten by incoming new ones. Samples can then be drawn uniformly (pseudo-)randomly from $\mathcal{B}$.

We mentioned above that experience replay addresses the problem of correlation among successive environmental observations, which may be beneficial. Another important consequence is that we may collect multiple quadruples simultaneously in a so-called batch. We can then make training more stable by averaging over the update directions suggested by the quadruples of the batch. This idea is what underlies mini-batch gradient descent, which lies between the two extremes of pure stochastic gradient descent (where we effectively use a batch size of 1) and classic, full-dataset parameter updating (which is often unwieldy for modern, large-scale datasets). Mathematically, mini-batch updating changes Equation 19 to

$$\begin{aligned} \theta_{t+1} = \theta_t + \frac{\alpha}{|B|} \sum_{t' \in B} \Big(& \\ & r_{t'+1} + \gamma \max_{a \in \mathcal{A}} Q\Big(\text{F}(\mathbf{s}_{t'+1}), a; \overline{\theta}_t\Big) - Q(\text{F}(\mathbf{s}_{t'}), a_{t'}; \theta_t) \Big) \\ & \qquad \cdot \nabla_{\theta_t} Q(\text{F}(\mathbf{s}_{t'}), a_{t'}; \theta_t), \end{aligned} \tag{20}$$

where $B \in [\min(0, t - B_{\text{replay}}), \min(0, t-1)]^{B_{\text{batch}}}$ of size $B_{\text{batch}} \geq 1$ is a batch of chosen quadruple time steps, and where the $t' \in B$ emphasise that the current time step t almost always differs from the time steps at which B's quadruples were drawn. Mnih and colleagues used such mini-batch training, with B_{batch} set to 32.

Atari 2600 environment. With the DQN and experience replay buffer specified, we only need to discuss the environment used by Mnih and colleagues. They have chosen to test their system in the Atari Learning Environment (ALE; Bellemare, Naddaf, & Bowling, 2013), a piece of emulation software that allows one to programatically interact with video games from the Atari 2600 console. Although an old

console—the system was brought to market in 1977 (Jain et al., 2011)—its games still are challenging to humans.

Each ALE environment consists of a game emulator that, at every time step, provides an observed game frame $[0, 255]^{210\times160\times3}$. The emulator runs at 60 frames per second, but does not run in real-time; if the DQN agent requires more than $1/60^{\text{th}}$ of a second, the emulator will simply wait for the agent's response. Since the games are so diverse, the action space $\mathcal{A}$ varies per game. For instance, *Pong* provides the integer space $\mathcal{A}_{Pong} = [0, 5]$ which corresponds to NO-OP, firing, moving left or right, and firing *and* moving left or right. Meanwhile, the legal action set for *Ms. Pac-Man* is $\mathcal{A}_{Ms.\ Pac\text{-}Man} = [0, 8]$ which excludes the option to fire, but includes the options to move diagonally in all four directions. A similar principle applies to the reward spaces: $\mathcal{R}_{Pong} = \{-1, 0, 1\}$ (for losing, neither losing or winning, and winning, respectively) while $\mathcal{R}_{Ms.\ Pac\text{-}Man} = \{10, 50, \ldots, 1{,}600\}$ (for various types of 'pellets', eating the ghosts, and losing lives).

The DQN authors make a few modifications to the basic environments provided by the ALE to enable faster learning and discourage learning complete trajectories, which would not count as true learning. First, they let the agent perform the NO-OP action uniformly pseudo-randomly between 1 to 30 times at the start of each episode. Second, when a life is lost in games which support lives, this is viewed as a game-over: it ends the episode. Since separate lives frequently have similar trajectories, separating them in episodes enables the DQN to collect more experience replay quadruples for 'true' episodes, and is thus arguably more informative than not ending on life losses. Further, Mnih et al. only take a new action every fourth frame of the game, repeating action input between these 'decision time steps', in order to reduce computational demand by around 75%, while not significantly impacting the gameplay. The only exception is *Space Invaders*, as certain blinking objects in the observations would be lost with a skip size of four; there, a skip size of three is used instead. Lastly, rewards are transformed as follows: if a reward is negative, it is set to -1; if it is zero, it remains 0, and if it is positive, it is set to 1. An obvious caveat with this procedure is that information from magnitude in reward signal is lost—think, for instance, of the various reward types in Ms. Pac-Man. Mnih and colleagues argue that the modification has a net benefit as it eases parameter configuration across the games that they tested, which were 7 and 49 in Mnih et al. (2013) and Mnih et al. (2015), respectively.

Finally, we briefly touch on the preprocessing used in order to feed observations to the DQN; in Equations 19–20 this preprocessing is implemented via the former feature extractor, F. First, the input image is entry-wise maximised with the directly preceding frame, which sometimes is referred to as 'max(imum)-pooling'. Then, this pooled frame is sent to the greyscale (luminance) colour space. Afterwards, it is shrunk in height and width to the size $84 \times 84 \times 1$ pixels.

We remark on an important detail here. The paper states that "[the authors] take the maximum value for each pixel colour value over the frame being encoded and the previous frame" (p. 534), and that afterwards the luminance is taken of this pooled frame. In practice, this actually may have been the other way around, as there is no clear definition of taking the maximum of two colours. Of course, it is possible to take the entry-wise maximum of the red, green, and blue channels and thereby define a maximisation operation, but this is not explicitly defined in the paper.

Note that this operation is a very minor form of manual intervention, while the authors stress DQN's independence in extracting relevant information from the observations. In our experimentation, which we will discuss later in more detail, we found that not cropping in this manner did not lead to any noticeable performance degradation.

Rainbow

Introduced three years later, Rainbow (Hessel et al., 2018) improves considerably on the Nature DQN agent. The authors achieve this feat by fusing multiple extensions suggested in the literature into a single updated design, specifically choosing extensions that do not interfere with or contradict one another. In total, six such extensions are used, which we will now briefly go over. In the following, we write updates to parameters as if a single-entry mini-batch with index t' is used, in order to keep notation more concise; via Equation 20, one can obtain analogous mini-batch updates.

First is the idea to decouple DQN's action-selection from the action-value update in which it is embedded; this is known as double Q-learning (van Hasselt, 2010). Specifically, we perform the decoupling by splitting the action-value function into two: one function for the selection task, another for evaluation. This has the benefit of reducing positive bias caused by the bootstrapping nature of Q-learning. In our case, we can use the separate parameterisations θ_t and $\overline{\theta}_t$ to achieve the intended goal. Given the original DQN update (Equation 19), we may adapt it as follows:

$$\theta_{t+1} = \theta_{t'} + \alpha \cdot \Big(r_{t'+1} + \gamma\, Q\Big(\mathrm{F}(\mathbf{s}_{t'+1}),\ \arg\max_{a\in\mathcal{A}} Q(\mathrm{F}(\mathbf{s}_{t'+1}), a; \theta_{t'});\ \overline{\theta}_t\Big) - Q(\mathrm{F}(\mathbf{s}_{t'}), a_{t'}; \theta_t)\Big) \cdot \nabla_{\theta_t} Q(\mathrm{F}(\mathbf{s}_{t'}), a_{t'}; \theta_t) \tag{21}$$

Notice that this change technically makes the update use semi-gradients instead of 'true' gradients, as we do not account for the θ_t in the action-selection argument maximum.

The second extension builds on top of the first: it restructures the DDQN architecture into a dueling architecture

(Z. Wang et al., 2016). Essentially, the dueling architecture leaves the convolutional layers of the original Nature DQN architecture as-is, while replacing the last two feed-forward layers by two separate streams of fully-connected layers. Let $C(\text{F}(\mathbf{s}_t); \theta_t)$ be the result of sending preprocessed state $\text{F}(\mathbf{s}_t)$ through only the convolution layers. Then one stream estimates the state-value function output $V(\, C(\text{F}(\mathbf{s}_t); \theta_t);\, \theta_t\,)$ while the second stream predicts so-called state-action advantages $A(\, C(\text{F}(\mathbf{s}_t); \theta_t),\, a;\, \theta_t\,)$, for all $a \in \mathcal{A}$. An advantage, intuitively, is the excess in cumulative expected future-discounted reward when choosing a specific action in a state, in comparison to the 'action-average' reward sum in said state. Mathematically put, $A(\mathbf{s}_t, a_t) \in \mathbb{R}$ (for any $t \in \mathcal{T}$) would normally be defined as

$$A(\mathbf{s}_t, a_t) = Q(\mathbf{s}_t, a_t) - V(\mathbf{s}_t). \tag{22}$$

The state-value stream has a single, real-valued scalar output, while the advantage stream has $|\mathcal{A}|$ outputs.[4] Then, the dueling architecture computes the action-value function output according to

$$\begin{aligned} Q(\text{F}(\mathbf{s}_t), a_t; \theta_t) = {} & V(\, C(\text{F}(\mathbf{s}_t; \theta_t));\, \theta_t\,) \\ & + A(\, C(\text{F}(\mathbf{s}_t); \theta_t),\, a_t;\, \theta_t\,) \\ & - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} A(\, C(\text{F}(\mathbf{s}_t); \theta_t),\, a;\, \theta_t\,). \end{aligned} \tag{23}$$

The main benefit of this refactoring is that the resultant network can learn the value of states without having to have explored all actions within said state. This is especially useful in 'liminal states' in which taking any particular action has no dramatic effect on agent performance.

The third extension used in Rainbow is to use multi-step temporal difference targets (Watkins, 1989). Recall Equation 15, in which we already introduced multi-step time difference action-value updates. If we copy the first summand of the parenthesised expression following the α scalar, we almost already have the desired multi-step variant of our TD target. What remains is replacing $Q_{t+m-1}(\mathbf{s}_{t+m}, a_{t+m})$ by

$$Q\Big(\, \text{F}(\mathbf{s}_{t'+m}),\ \arg\max_{a \in \mathcal{A}} Q(\text{F}(\mathbf{s}_{t'+m}), a; \theta_t);\ \overline{\theta}_t \,\Big),$$

and also replacing each preceding reward t by a t' counterpart to emphasise we draw samples from the experience replay buffer. If the resulting TD target replaces its single-step counterpart in Equation 19, we arrive at a multi-step variant of DQN parameter updating. The last step is to adapt this update for the dueling network architecture, but that can be achieved by using Equation 23's replacement for the original Q-values.

The fourth extension is perhaps most technical of all six. It involves, in essence, replacing each output node $Q(\mathbf{s}_t, a_t; \theta_t)$ by multiple nodes that together form a (predicted) probability distribution over the possible returns, given that action a_t is taken in state $\mathbf{s}_t$ (Bellemare, Dabney, & Munos, 2017). Since probability distributions provide richer output, this alteration may both improve human understanding of the model and improve agent performance.

Let us make this more precise. The dueling network with multi-step TD targets that we currently have provides, for each state-action pair, an estimation of its action-value: $Q(\mathbf{s}_t, a_t; \theta_t)$. Now introduce three hyper-parameters: $N_{\text{atoms}} \geq 1$, $G_{\min}, G_{\max} \in \mathbb{R}$. Together, they specify a (heavily discretised) sample space Ω_{returns} for reinforcement learning returns:

$$\Omega_{\text{returns}} = \left\{ G_{\min} + (i-1) \cdot \frac{G_{\max} - G_{\min}}{N_{\text{atoms}}} \,\middle|\, i \in [1, N_{\text{atoms}}] \right\}. \tag{24}$$

We are now in the position to replace our $Q(\mathbf{s}_t, a_t; \theta_t)$-predictions by functions that assign probability mass to each of these atoms, depending on the specific state-action pair under consideration. Denote these functions by $\mathbf{p}(\mathbf{s}_t, a_t; \theta_t) \in [0,1]^{N_{\text{atoms}}}$. Then let $d_t = (\Omega_{\text{returns}}, \mathbf{p}(\text{F}(\mathbf{s}_t), a_t; \theta_t))$ be the network's predicted probability distribution.

What is still required is a way to train these probability mass assignment functions. Bellemare and colleagues suggest to leverage the analogy of the Bellman equation to relate the predicted probability distribution of return at time step t to its target counterpart at $t+m$, albeit with the latter's support scaled by γ^m and shifted by $r_{t+1} + \ldots + \gamma^{m-1} r_{t+m}$. That is, let

$$\begin{aligned} d_{\text{target};\, t+m} = \Big(& r_{t+1} + \cdots + \gamma^{m-1} r_{t+m} + \gamma^m\, \Omega_{\text{returns}}, \\ & \mathbf{p}\Big(\text{F}(\mathbf{s}_{t+m}),\ \arg\max_{a \in \mathcal{A}} Q(\text{F}(\mathbf{s}_{t+m}), a;\, \overline{\theta}_t\Big)\Big) \end{aligned} \tag{25}$$

be this target distribution, with $m \geq 1$ indicating as before the number of time steps to consider the value over.

This discrepancy in distributional support is immediately the next challenge: in order to meaningfully compare the two distributions, they need to have identical support. Thus, Bellemare et al. project the G_{t+1} distribution's support space onto that of G_t by means of the L^2-projection $\Phi_{\Omega_{\text{returns}}}$ (Bellemare et al., 2017, p. 454) and can subsequently use the Kullback-Leibler divergence as an error signal between d and d_{target}.

Hessel et al. implement this distributional prediction of action-values at the end of their Rainbow system, noting that target weights $\overline{\theta}_t$ are kept frozen, as was the case in the Nature DQN architecture. In order to guarantee the outputs can be

[4] In Z. Wang et al. (2016), separate symbols θ, β, and α were used to refer to the parameters of the pre-stream convolutions, the state-value function, and the advantage function, respectively. For notational simplicity, we use the θ_t hyperparameter in all three; it covers the parameterisation for the complete dueling architecture in our text. This may incorrectly convey that the components all require the full set of parameters θ_t; this is not our intention.

interpreted as probability masses, they also apply the SoftMax operator across every $Q(\text{F}(\mathbf{s}_t), a; \theta_t)$ predicted probability distribution, for all $a \in \mathcal{A}$.

Fifth, instead of using regular experience replay as outlined above, Hessel and colleagues use prioritised experience replay instead (Schaul, Quan, Antonoglou, & Silver, 2016). With prioritised experience replay, quadruples in the experience replay buffer $\mathcal{B}$ get assigned a probability of selection based, intuitively, on the degree to which the network was off in its prediction for the sample: the greater the discrepancy, the higher the assigned probability. Since the Kullback-Leibler divergence is used to train the network, this value is used to determine, up to a scalar, the probability of selection.

Last but not least, the sixth extension involves 'noisifying' the fully-connected layers of the network following the approach taken by Fortunato et al. (2018). Although in the original Nature DQN these would have been the last two layers, they have been replaced by the dueling architecture. However, since its two streams also consist completely of fully-connected layers, those layers are subjected to the noisification instead.

Noisification of a fully-connected layer consists of linearly adding a (pseudo-)randomly scaled bias vector and -weights matrix. Let $\mathbf{x} \in \mathbb{R}^{N_{\text{in}}}$ denote the input and $\mathbf{y} \in \mathbb{R}^{N_{\text{out}}}$ the output, let $\mathbf{b} \in \mathbb{R}^{N_{\text{out}}}$ and $W \in \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ be the regular bias and weight matrix, and let $\mathbf{b}_{\text{noise}} \in \mathbb{R}^{N_{\text{out}}}$ $W_{\text{noise}} \in \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ denote noisy bias and weight matrix counterparts. Further, let $\mathbf{e}_{\mathbf{b}} \in \mathbb{R}^{N_{\text{out}}}$, $\mathbf{e}_W \in \mathbb{R}^{N_{\text{out}} \times N_{\text{in}}}$ denote randomly-drawn noise scalars for the current time step. Then the output of the noisified, fully-connected layer is defined as

$$\mathbf{y} = \mathbf{b} + W\mathbf{x} + \mathbf{b}_{\text{noise}} \odot \mathbf{e}_{\mathbf{b}} + (W_{\text{noise}} \odot \mathbf{e}_{\text{W}})\,\mathbf{x},$$

where $\odot$ denotes the entry-wise multiplication. All the weights and biases are trainable while the $\mathbf{e}_{\mathbf{b}}$ and $\mathbf{e}_W$ are not; the latter could be regarded as part of the environment.

Besides the regularising effect that noisy fully-connected layers have, they have a specific application in DRL as well: they facilitate asymmetric exploration of the environment, in the sense that certain, relatively well-known parts of the environment may already be accounted for by the agent in terms of noise, leading to (near-)greedy behaviour there, while in other parts the agent may still explore due to the noise there not yet being controlled by the agent's parameterisation (Fortunato et al., 2018; Hessel et al., 2018).

Finally, we remark that the MOREL architecture is shown in Figure 1.

Object-sensitive deep reinforcement learning

In their 2017 paper, Li et al. suggest a new approach to interpretable DRL named object-sensitive deep reinforcement learning (O-DRL).

O-DRL proposes to explicitly include a notion of 'objects' in existing DRL models. Here, the authors do not specify what counts as an object. Instead, they appeal to the categories we as humans naturally use. For example, within a video game, human players naturally group sets of pixels and assign these high-level labels such as 'playable character', 'enemy', and 'collectible'. By providing a DRL model with such extra, high-level 'domain knowledge', said model may learn more efficiently.

Two technical ideas take an arguably prominent role in the paper—both directly related to the use of objects. As both will become relevant later in the thesis, we treat them in detail.

First, we take a close look at how objects are detected from raw environmental observations. Note that, as this requires us to introduce various computer vision concepts, we need to make a slight departure from the usual context of (deep) reinforcement learning.

Second, we move on to so-called saliency maps (OSMS); these bring us back again to the DRL context.

Template matching. It is natural to ask how Li and colleagues extract objects from raw environmental observations. To this end, they use template matching, a traditional computer vision technique. Simply put, given an object channel, a human expert selects a (relatively small) image representative of the object and supplies it to the template matching algorithm. This algorithm then sweeps from the top-left to the bottom-right over the raw image with the template, computing at each location a similarity score. If this score surpasses a threshold—again set by the human expert—we say that that position in the raw image contains the object. By performing this comparison at every raw image location, we obtain a 2D map of object locations; this map is the channel for the object under consideration.

Since template matching will be used in some of our experiments, it may be helpful to introduce this idea precisely. Let $H_{\text{template}}, W_{\text{template}} \in \mathbb{Z}^+$ denote the height and width of the template, and let $H_{\text{image}}, W_{\text{image}} \in \mathbb{Z}^+$ denote the same for the image to template-match. We require that

$$H_{\text{template}} \leq H_{\text{image}} \text{ and}$$
$$W_{\text{template}} \leq W_{\text{image}}$$

in order to avoid problems pertaining to out-of-bounds matching of templates. Further, introduce

$$I_{\text{template}} \in [0, 255]^{H_{\text{template}} \times W_{\text{template}} \times C} \text{ and}$$
$$I_{\text{image}} \in [0, 255]^{H_{\text{image}} \times W_{\text{image}} \times C}$$

to represent the template and image, respectively. Here, $C \geq 1$ is the number of channels; commonly, $C \in \{1, 3, 4\}$ for greyscale, RGB, or RGBA images, respectively. Then a

similarity-scoring function M_{sim} can be defined as follows:

$$M^{\text{M}}_{\text{sim}}\Big(I_{\text{image}}, I_{\text{template}}\Big)_{y,\,x} = \sum_{\substack{y' \in [0,\, H_{\text{sim}}\rangle,\\ x' \in [0,\, W_{\text{sim}}\rangle,\\ c \in [0,\, C\rangle}} \text{M}\Big(I_{\text{image};\, y',\, x',\, c},\; I_{\text{template};\, y+y',\, x+x',\, c}\Big), \quad (26)$$

where $H_{\text{sim}} = H_{\text{image}} - H_{\text{template}} + 1$ and
$W_{\text{sim}} = W_{\text{image}} - W_{\text{template}} + 1$,

and where the implementation of M_{sim} depends on the algorithm M that is used, not to be confused with the previously-used time step range, m. The implementation that is most relevant within this thesis is the square difference method,

$$\text{M}_{\text{SD}}\Big(p_{\text{image}}, p_{\text{template}}\Big) = \Big(p_{\text{template}} - p_{\text{image}}\Big)^2. \quad (27)$$

This is, however, not the method that Li et al. used. Instead, they worked with the correlation coefficient method, M_{CC}, due to its balancing of accuracy with computational speed. Instead of accepting I_{image} and I_{template} directly, $M^{\text{M}_{\text{CC}}}_{\text{sim}}$ works with modified versions of them, defined as follows:

$$I'_{\text{template};\, y',\, x',\, c} = I_{\text{image};\, y',\, x',\, c} - \frac{\sum_{y'' \in [0,\, H_{\text{template}}\rangle,\; x'' \in [0,\, W_{\text{template}}\rangle} I_{\text{template};\, y'',\, x'',\, c}}{H_{\text{template}} \cdot W_{\text{template}}}, \quad (28)$$

$$I'_{\text{image};\, y+y',\, x+x',\, c} = I_{\text{image};\, y+y',\, x+x',\, c} - \frac{\sum_{y'' \in [0, H_{\text{template}}\rangle,\; x'' \in [0, W_{\text{template}}\rangle} I_{\text{image};\, \text{y + y''},\, \text{x + x''},\, \text{c}}}{H_{\text{template}} \cdot W_{\text{template}}}. \quad (29)$$

Given these, M_{CC} is simply given by

$$\text{M}_{\text{CC}}\Big(p_{\text{image}}, p_{\text{template}}\Big) = \Big(p_{\text{template}} \cdot p_{\text{image}}\Big). \quad (30)$$

We would like to discuss two extensions to basic template matching, as discussed just now. The first addition is to normalise the similarity scores in $M^{\text{M}}_{\text{sim}}$. This produces a normalised scores map, $\widetilde{M}^{\text{M}}_{\text{sim}}$, defined as

$$\widetilde{M}^{\text{M}}_{\text{sim}}\Big(I_{\text{image}}, I_{\text{template}}\Big) = \frac{M^{\text{M}}_{\text{sim}}\Big(I_{\text{image}}, I_{\text{template}}\Big)}{\sqrt{\left(\sum_{\substack{y' \in [0, H_{\text{template}}\rangle,\\ x' \in [0, W_{\text{template}}\rangle,\\ c \in [0, C\rangle}} I^2_{\text{template};\; y',\, x',\, c}\right)\left(\sum_{\substack{y' \in [0, H_{\text{template}}\rangle,\\ x' \in [0, W_{\text{template}}\rangle,\\ c \in [0, C\rangle}} I^2_{\text{image};\; y+y',\, x+x',\, c}\right)}}. \quad (31)$$

Besides using M_{CC}, Li et al. normalise the similarity scoring maps according to the equation above in order to enable easier thresholding on potentially detected objects. As will be shown later, we use this idea as well. The second extension to basic template matching is to use masks. With masks, a subset of positions $P_{\text{masked}} \subseteq [0, H_{\text{template}}\rangle \times [0, W_{\text{template}}\rangle$, with $P_{\text{masked}} \neq [0, H_{\text{template}}\rangle \times [0, W_{\text{template}}\rangle$, is ignored or 'masked' when comparing I_{template} to I_{image}; only positions outside P_{masked} are considered. Mathematically, adopting masks leads to a replacement of the vertical and horizontal iteration ranges under the summations in Equations 26–31: (y', x')-pairs are now drawn from $[0, H_{\text{template}}\rangle \times [0, W_{\text{template}}\rangle \setminus P_{\text{masked}}$, instead. Additionally, the denominators in Equations 28 and 29 are replaced by $H_{\text{template}} \cdot W_{\text{template}} - |P_{\text{masked}}|$.

Object saliency maps. Increased learning efficiency is not the only potential benefit of O-DRL. The design of object channels was also motivated in part by enabling more intuitive interpretation of model input. As such, we can leverage these channels to provide insight into model decision-making. In order to do so, the authors introduce the object saliency map, a graphic that shows per object in the state-action pair under consideration how much said object contributes positively or negatively to the overall action value. To understand object saliency maps, however, we first need to understand 'regular' saliency maps.

A saliency map (Simonyan, Vedaldi, & Zisserman, 2013) can be computed for any feed-forward neural network, for any legal input. Consider any hidden or output neuron in the network. Then that neuron's activation can be viewed as a highly non-linear function of the network's input. Choose any state $\mathbf{s} \in \mathcal{S}$ to serve as an observation that is input to the network, ignoring the time step subscript. Furthermore, let $\text{a}(\mathbf{s}) \in \mathbb{R}$ denote the activation of the neuron under consideration when supplying $\mathbf{s}$ at the network's input. (Do not confuse $\text{a}(\mathbf{s})$ with a, the symbol for an arbitrary action in $\mathcal{A}$.) Since $\text{a}(\mathbf{s})$ is highly non-linear, it is difficult to understand how the entries in $\mathbf{s}$ affect $\text{a}(\mathbf{s})$. At the cost of only approximating the answer, we can rewrite $\text{a}(\mathbf{s})$ as a Taylor expansion:

$$\begin{aligned}\text{a}(\mathbf{s}) = {} & \text{a}(\mathbf{s}_0) \\ & + \Big(\nabla_{\mathbf{s}} \text{a}(\mathbf{s})\,\Big|_{\mathbf{s}=\mathbf{s}_0}\Big)(\mathbf{s} - \mathbf{s}_0) \\ & + H(\mathbf{s}_0)\,(\mathbf{s} - \mathbf{s}_0)^2 \\ & + \cdots\end{aligned} \quad (32)$$

where $\mathbf{s}_0$ is the input around which we want to approximate. Simonyan and colleagues choose $\mathbf{s}_0 = \mathbf{0}$ and limit the polynomial to the first degree, thereby obtaining the linear approximation

$$\text{a}(\mathbf{s}) \approx \text{a}(\mathbf{0}) + \Big(\nabla_{\mathbf{s}} \text{a}(\mathbf{s})\,\Big|_{\mathbf{s}=\mathbf{0}}\Big) \cdot \mathbf{s}. \quad (33)$$

Of primary interest in Equation 33 is the gradient term $\nabla_{\mathbf{s}} \text{a}(\mathbf{s})\,|_{\mathbf{s}=\mathbf{0}}$, because each entry in it can be seen as functioning as a weight for a corresponding pixel channel brightness in the state $\mathbf{s}$: the greater the weight, the less the associated pixel needs to change in channel brightness in order to affect $\text{a}(\mathbf{s})$. In this way, we can order the pixels in $\mathbf{s}$ based on their 'importance' in establishing the activation $\text{a}(\mathbf{s})$ of $\mathbf{s}$: those

entries with relatively high weights are relatively 'important' or 'salient' to the network. Thus, Simonyan and colleagues call this vector the saliency map of the neuron under consideration.

Li and colleagues build on the idea of saliency maps to obtain object saliency maps (OSMS). Since obtaining the partial derivatives of $a(\mathbf{s})$ with respect to objects is relatively difficult in comparison to taking these derivatives with respect to simply all input pixels—the entries of $\mathbf{s}$—Li et al. instead propose the following: for each object for which we would like to determine the partial derivative, we create two states: $\mathbf{s}_{\text{present}}$ and $\mathbf{s}_{\text{absent}}$. The former state is simply the original, unmodified input. The latter state is identical to $\mathbf{s}_{\text{present}}$, except that the object under consideration has been removed from all channels—both the raw input image channels and the relevant object channel.[5] By calculating

$$a(\mathbf{s}_{\text{present}}) - a(\mathbf{s}_{\text{absent}}) \tag{34}$$

we may obtain an approximate measure of the contribution that the object under consideration makes to $a(\mathbf{s}_{\text{present}})$. This computation is both mathematically simple and affords a straightforward interpretation: if $a(\mathbf{s}_{\text{present}}) > a(\mathbf{s}_{\text{absent}})$, then the object contributes positively to the activation total; vice-versa if $a(\mathbf{s}_{\text{absent}})$ has a greater activation.

Importantly, the just-explained mode of computing saliencies assumes that objects contribute to state $\mathbf{s}_{\text{present}}$'s total saliency strictly on their own. Put differently, there should be no interactions among objects in contributing to total saliency. This assumption is not stated explicitly in Li et al. (2017).

The authors continue by evaluating O-DRL using object-augmented versions of the original Nature DQN, discussed above, on Double DQNs (DDQN (van Hasselt, 2010), Dueling DQNs (Z. Wang et al., 2016) and 'Advanced [sic] Actor-Critic' (A3C, actually known as Asynchronous Advantage Actor-Critic; Mnih et al., 2016). They test their models on five Atari 2600 video games from the OpenAI Gym suite (Brockman et al., 2016): *Freeway*, *River Raid*, *Space Invaders*, *Bank Heist*, and *Ms. Pac-Man*. These have specifically been chosen because of their frequent appeal to objects. Put briefly, for every one of these games, one of the object-augmented DRL methods obtained the best average score out of all models that were considered for that game, demonstrating that introducing objects at the input may be an effective approach to improve DRL performance. Additionally, the authors show situations in which O-DRL's object saliency maps may improve interpretation of model decisions relative to using Simonyan et al.'s original saliency maps approach.

Although Li and colleagues show promising results, there are three challenges to adopting this approach widely within DRL. First, O-DRL requires us to provide object templates up front. Many environments change continually, and as such make the use of predefined templates a brittle choice for a model. Additionally, O-DRL assumes implicitly that the object templates 'work': they match when objects under consideration truly appear on-screen, and they do not when no such object is present. To this end, Li et al. test their templates against human annotations on a set of pre-collected video game frames, but such testing becomes nearly impossible for the aforementioned more complex situations that DRL may be deployed in, such as real-time car driving. Lastly, template matching can become costly when the input is of a high resolution, when many frames need to be evaluated, or when many different object channels are introduced, or need to be introduced, as is the case in more complex environments (for instance, modern video games).

Since we concentrate on direct extensions of the Nature DQN architecture, and since Li et al. included an object-oriented variant for it in their work, we have included it in our architectures diagram; see Figure 1.

Motion-oriented reinforcement learning

One year after the presentation of O-DRL, Goel et al. (2018) introduce Motion-Oriented Reinforcement Learning (MOREL). In a sense, MOREL can be viewed as an improved and more generalised version of O-DRL, although it requires the agent's deep learning model to change—something that was expressly not needed in O-DRL, because there only the input was adapted.

In essence, Goel et al. propose to change the network in such a way, so that it extracts features from the environment that closely relate to a human's notion of objects. This is achieved by adapting a deep learning model used originally to compute optical flow. Before explaining the adapted model, it may be helpful to give a minimal explanation of what optical flow is; the design of the model may become more apparent afterwards.

Optical flow. Let there be two images,

$$I_1, I_2 \in [0, 255]^{H_{\text{images}} \times W_{\text{images}} \times C_{\text{images}}},$$

where $H_{\text{images}}, W_{\text{images}}, C_{\text{images}} \geq 1$. Image I_1 precedes I_2 in a temporal sense. For example, I_1 and I_2 could be a pair of successive frames in an Atari 2600 game. Intuitively, if spatial displacements of both the observer and the contents of the observed scene are not too great, then it may be plausible to relate I_1 and I_2 to one another as follows:

$$I_{1;\,y,\,x} = I_{2;\,y+\Delta y,\,x+\Delta x}, \tag{35}$$

where $(y, x) \in [0, H_{\text{images}}\rangle \times [0, W_{\text{images}}\rangle$ and where $\Delta y, \Delta x \in \mathbb{R}$. Equation 35 is known as the brightness constancy as-

[5] Importantly, by 'object' we mean single objects, and not single categories of objects. Hence, if a raw input image contains multiple instances of some object category, we remove these objects individually, and not all of them together, simultaneously. This allows us to determine the contribution to activation that single objects make separately.

sumption[6], because it assumes that pixel intensities at any position (y, x) will remain the same over the duration, taking into account small spatial motion $(\Delta y, \Delta x)$. Now, we can collect the pairs $(\Delta y, \Delta x)$ per pixel to obtain a map $(V, U) \in \mathbb{R}^{[0,H_{\text{images}}\rangle \times [0,W_{\text{images}}\rangle \times 2}$ which describes how pixels from image I_1 move to arrive at their positions in image I_2. This pair (V, U), better known as the optical flow between I_1 and I_2, is sometimes defined as "[...] the distribution of apparent velocities of movement of brightness patterns in an image", after Horn and Schunk (1981).

Because the brightness constancy assumption directly relates I_1 to I_2 via the optical flow, the assumption may naturally be used to construct various loss measures for evaluating estimates of optical flow between I_1 and I_2. One example is

$$\mathcal{L}_{\text{rec}}(I_1, I_2, (V', U')) = \sum_{\substack{y \in [0,H_{\text{images}}\rangle, \\ x \in [0,W_{\text{images}}\rangle}} \left\| I_{1;\,y,\,x} - I_{2;\,y+V'_{y,\,x},\,x+U'_{y,\,x}} \right\|_1 , \tag{36}$$

with $(V', U') \in \mathbb{R}^{[0,H_{\text{images}}\rangle \times [0,W_{\text{images}}\rangle \times 2}$ being the predicted optical flow. The loss expressed in Equation 36 is used by Vijayanarasimhan, Ricco, Schmid, Sukthankar, and Fragkiadaki (2017). The authors of MOREL instead use a more sophisticated loss—structural dissimilarity (DSSIM), introduced by Z. Wang, Bovik, Sheikh, and Simoncelli (2004)—because its loss signal also considers relatively spatially distant reconstruction errors (Goel et al., 2018, p. 5,686). Regardless, the reconstruction loss that was just introduced leads to a straightforward idea: design a deep neural network, if given a pair (I_1, I_2), returns an optical flow, which we then evaluate using DSSIM to train the model. This is indeed what Goel and colleagues did, and it is central to how MOREL works.

MOREL network architecture. Before discussing the model, we should address an obvious question: how precisely does the prediction of optical flow relate to the detection of objects? In essence, the idea is to predict optical flow by letting the network internally estimate how a set $K \geq 1$ of objects will move from their positions in I_1 to those in I_2, separately. This is achieved by two network components: one estimates per each pixel how probable that pixel is to belong to object k, for each $k \in [0, K\rangle$, producing a 2D map $P_k \in [0, 1]^{H_{\text{images}} \times W_{\text{images}}}$; another estimates the vertical and horizontal displacements $(v'_k, u'_k) \in \mathbb{R}^2$ of each of the k objects, plus the movement of the camera $(v'_{\text{cam}}, u'_{\text{cam}}) \in \mathbb{R}^2$. Then, naturally,

$$(V', U') = \left(\sum_{k=0}^{K-1} (v'_k, u'_k) \right) - (v'_{\text{cam}}, u'_{\text{cam}}). \tag{37}$$

This optical flow estimate can subsequently be supplied to the aforementioned loss, and the error it produces may then be used for backpropagation throughout the layers of the network. Notice that, of the two components, we are most interested in the part that produces the P_ks as it can be viewed as segmenting the objects from the image pair.

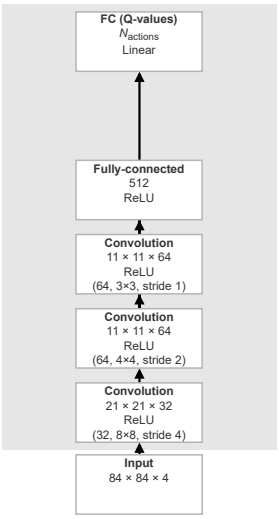
The network used in Goel et al.'s implementation has been derived from an earlier work that considered the problem of predicting I_2 from I_1 in isolation, apart from the DRL setting (Vijayanarasimhan et al., 2017). The original model, known as the Structure-from-Motion Network (SfM-Net) included an extra subnetwork known as the structure network, meant for predicting per-pixel depth in the image pairs. Since the SfM-Net authors considered general image pairs—notably, video frames captured by a vehicle driving around in the real world—the inclusion of such a depth-predicting component made sense, as it allows for three-dimensional motion prediction. However, since Goel et al. applied SfM-Net in the two-dimensional problem domain of Atari 2600 games, they removed the structure network.

MOREL's architecture is visually summarised in Figure 1d. Immediately observable is that the network actually consists of two smaller networks: one computes optical flow for moving objects, and, in doing so, derives segmentations for moving objects ('dynamic net', grey block on the left); the other addresses the remainder of the environmental observations, namely all that is static ('static net', grey block on the right). Both subnetworks share the same architecture for downsampling the input, although the parameterisation of them may differ. As could be expected from our discussion of optical flow, the input consists of two single, successive environmental observations stacked depth-wise. This stacked input is primarily meant for the dynamic objects subnetwork, but since, at least in principle, it should not affect the static object network, this stack is fed to both subnetworks simultaneously.

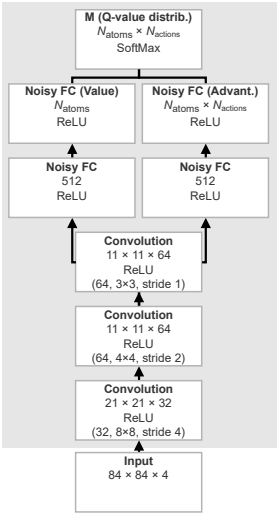
Let us first consider the dynamic objects subnetwork in more detail. After three layers of downsampling by convolutions, the subnetwork splits into two branches: one is designed to estimate the k object segmentation masks P_k while the other estimates the movements (v'_k, u'_k) of said objects, plus the movement $(v'_{\text{cam}}, u'_{\text{cam}})$ of the camera. Hence, these two branches implement in a neural network what we discussed above mathematically. Finally, the two branches are merged again in the block named 'Optical Flow' in Figure 1d, and it performs a bilinear interpolation that uses I_2 and (V', U') to reconstruct I_1. Subsequently, DSSIM can be used to derive the loss of Equation 36. Notably, it does not involve any trainable parameters.

The main reason why optical flow was introduced into Goel et al.'s paper was to improve existing DRL systems. (They used Advantage Actor-Critic [A2C; Baird, 1993] in their experimentation.) In the middle of the dynamic object

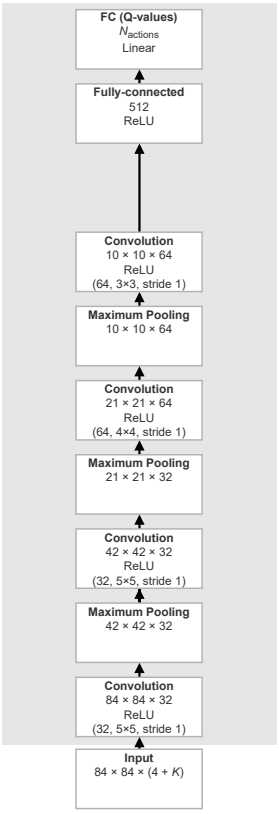
[6] For consistency, we always use the term 'brightness' in referring to the brightness constancy assumption, even though C_{images} may be greater than 1. One could alternatively argue for the term 'colour constancy assumption' if $C_{\text{images}} = 3$, and similarly so for other numbers of channels.



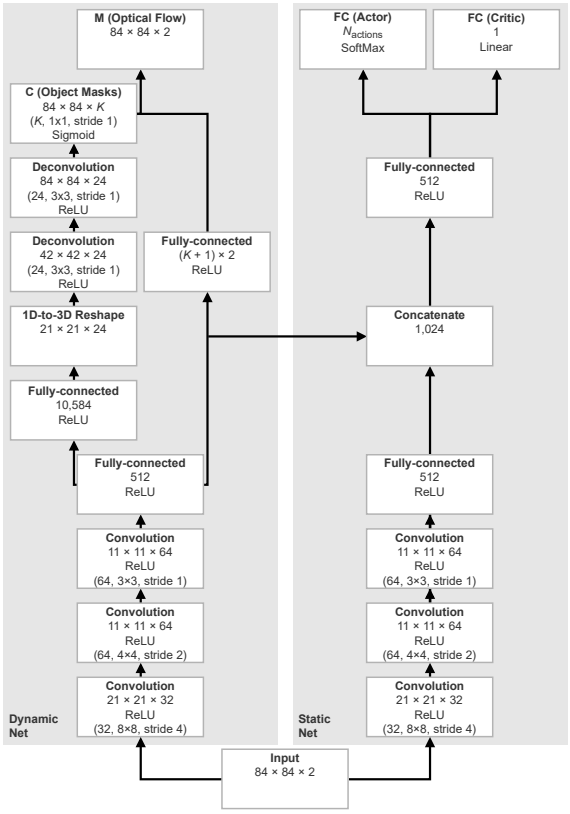
(a) The original Nature DQN (Mnih et al., 2015).



(b) Rainbow (Hessel et al., 2018).



(c) Li et al. (2017)'s adaptation of the Nature DQN.



(d) Goel et al. (2018)'s MOREL model.

Figure 1. Architecture diagrams of DRL models discussed in the Related Works. The shorthands C, FC, and M stand for convolution, fully-connected, and manually computed (that is, not using any well-known layer, but implementing something network-specific or unique), respectively. Further, N_{atoms} and N_{actions} refer to $|\mathcal{A}|$ and N_{atoms} within the text, and K is equivalent to Li et al. (2017) and Goel et al. (2018)'s number-of-objects hyper-parameter, K. Lastly, for Convolutions (shorthand: C), the second line of a block denotes the output size (without the batch dimension), while the fourth line denotes the triple (number of filters, kernel size, stride). The stride's first argument is the vertical displacement; the second argument represents the horizontal displacement. For the definitions of the activation functions Linear, Sigmoid, ReLU, and SoftMax, see Appendix A.

subnetwork a relatively low-dimensional layer is situated: the fully-connected layer of 512 entries. At any point in the complete network's state, this layer can be seen as encoding a latent representation of the moving objects, because it is used downstream in the computation of optical flow. Now, Goel and colleagues concatenate this low-dimensional middle layer with its 'twin' in the downsampling network of the static object network. The concatenated representation is in turn used by the critic and actor to predict $V(\mathbf{s})$ and $A(\mathbf{s}, a)$ values. In this manner, the entire system considers both influences from dynamic and static objects.

Training. Lastly, we comment on a decision made on MOREL's training regime. The network is not trained in its entirety starting from the very first episode. The authors observe that the dynamic object subnetwork's parameters can be initialised in such a way that the subnetwork is already strongly predisposed to detecting movement: by training it in isolation for a predefined number of time steps. Training observations can be collected in a straightforward manner by letting a fully random policy interact with the environment. Notice that the subnetwork thus learns in a fully-unsupervised manner: no reward signal is required, only inputs—that is, observations. Specifically, Goel and colleagues train the dynamic subnetwork for the first 250,000 time steps only on observations from this random policy. Note that since the static object subnetwork has not been connected yet during this timeframe, the cross-subnetwork connection that allows the concatenation between the aforementioned two fully-connected layers is absent as well.

Finally, we must note that although the reconstruction loss is the primary error signal to steer parameter updates of the dynamic object network, another regularisation term is used to actually complete the loss:

$$\mathcal{L}_{\text{total}}(I_1, I_2, (V', U')) = \mathcal{L}_{\text{rec}}(I_1, I_2, (V', U')) + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}}\left(\left\{ \left(P_k, v'_k, u'_k \right) \right\}_{k=0}^{K-1} \right) \quad (38)$$

where $\lambda_{\text{reg}} \in [0, 1]$ and $\mathcal{L}_{\text{reg}}$ is defined as

$$\mathcal{L}_{\text{reg}}\left(\left\{ \left(P_k, v'_k, u'_k \right) \right\}_{k=0}^{K-1} \right) = \sum_{k=0}^{K-1} \left\| P_k \left[(V')^\top ; (U')^\top \right] \right\|_1 \quad (39)$$

where the notation $[\,\cdot\,;\,\cdot\,]$ is meant to denote depth-wise concatenation. λ_{reg} is slowly linearly increased from 0 to 1 over the first 100,000 time steps to avoid premature, excessive penalisation of high values for entries in the P_k, (v'_k, u'_k) 'multiplications'.

We omit other training details as these are not directly relevant to our discussion. The interested reader is referred to Goel et al. (2018).

Methdology

We begin the Methodology by first operationalising our two research questions. Thereafter, we give an overview of the three experiments we plan to perform based on the operationalised research questions, after which we explain the procedures for these three experiments in detail in separate subsections.

Operationalisation. As we explained in the Introduction, we focus on two research questions within this thesis. Both need to be operationalised for further use in the Methodology.

We recall the first research question: "Does representing the state by its high-level objects accelerate learning in deep reinforcement learning methods?" Multiple components of the question require further specification.

To begin, we narrow 'state representation by high-level objects' down to mean within this thesis: a representation of state by means of either template-based object channels as proposed by Li et al. (2017), or via object segmentation masks computed by the dynamic subnetwork of Goel et al. (2018). We argue for qualifying the representations derived by both approaches as 'high-level', because both representation types have a clear correspondence to a human's notion of an 'object'.

Next, we must address what it means for a method to have 'its learning be accelerated by a state representation'. For this, we introduce a performance measure also used in other works (Machado et al., 2018; Castro et al., 2018): the episode-averaged future-undiscounted return (of a collection of episodes). If we compute such average undiscounted returns over multiple, evenly-spaced points[7] during training of an agent by a DRL method, then we may construct an 'average undiscounted return curve' built from these points. Further, if we choose a specific average undiscounted return and compare the points at which this return is reached—preferably, using multiple such sampled curves—we may determine that the method using the representation structurally sees its return curves reach this chosen return at an earlier point than the method without the representation. The proportion between the former and the latter's average required number of points may then be regarded as the 'learning acceleration' of using the high-level representation for the method.

Moving to our second research question, we formulated it as "Can high-level object representations make deep reinforcement learning methods more explainable?" Here, we re-use our operationalisation of 'high-level object representations'. We also employ Gilpin et al. (2018)'s definition of 'explainability' from the Introduction, where we specifically emphasise the part "[. . .] [models that] produce insights about the causes of their decisisons".

At this point, the second question arguably still needs specification in two regards. First, we need to establish how we measure a DRL model's explainability, with and without the use of high-level object representations. Second, we need a procedure to causally determine the relationship between

[7] These 'points' will later be made precise using the term 'iteration'.

(I) the use of said high-level object representations, and (II) obtaining a model that measurably is more explainable.

As already expressed in the Introduction, we answer the second research question tentatively. We do so due to constraints of time and topical scope. Instead, we follow Li et al. (2017)'s method to leverage object-level representations to generate OSMs, which may make DRL methods more explainable. We provide these OSMs systematically over a preset number of DRL methods, environments, and states, and additionally cover these critically in our Discussion. Our intent, then, is that these results may help in further study of OSMs as a path to model explainability.

Methodology overview. Now that we have made our two research questions sufficiently precise, we continue by describing three experiments that we conduct to answer the research questions. In all three, we structurally collect average undiscounted return curves and object saliency maps.

In experiment 1, we largely repeat Li et al. (2017)'s procedure, although we introduce some changes to align their methodology more closely to both the original work done by (Mnih et al., 2015) as well as the baselines used within this thesis (Castro et al., 2018).

In the above procedure, the environment's 'raw' greyscale input is supplied to the agent, accompanied by $K \geq 1$ object channels. Following this approach has an intuitive appeal: we 'augment' the input with additional information, which, we hope, may help in accelerating learning speed and making the agent more explainable. However, if the aforementioned two measures indeed appear to change from their non-object baselines, there is an obstacle hindering us from attributing said change to the addition of the K objects: since we also supplied the original (albeit greyscaled) screen to the agent, we cannot exclude with certainty that some information present on the 'raw' screens during the object-based experiments affected the deviation from the baselines, instead of the object channels.

To address this methodological problem, in experiment 2, we run experiments involving exclusively Li et al. (2017)-style object channels. This object channel-only model is the third, novel method to object-based DRL that we stated we would test in the Introduction. Since some environments provide a non-object background that may arguably be crucial to playing the game successfully—think for example of the maze layout in *Ms. Pac-Man*—we append to the K object channels a 'background' channel that exclusively encodes the background, without any objects within it. As such, it is different from the 'raw' screens that include both sources of information.

Then, in experiment 3, we shift attention away from Li et al. (2017)'s work and toward the model proposed by Goel et al. (2018). Here, Goel et al.'s model may be seen as an adaptation to Li et al.'s system, as both have similarly-structured object channels. The difference is that Goel et al. let these be computed 'autonomously' by the agent, requiring no manual involvement of a human supervisor. Like experiment 1, we change Goel et al. (2018)'s methodology in order to align it with the other work discussed and conducted in this thesis.

More specifically, we adapt MOREL's dynamic (or, 'unsupervised') subnetwork (schematically displayed in Figure 1d, left block in grey), as used in the pre-training stage of MOREL, to serve as a generator for motion mask channels that add to the 'raw' greyscale screens. As such, this last experiment is closer in form to experiment 1 than experiment 2, in the sense that it augments input instead of replacing input by object-only channels (plus a background channel), as is done in experiment 2.

As we argued in our motivation for experiment 2, using only object channels may give a better indication of the effect of using objects. Consequently, all else being equal, it may also be better to follow this approach with Goel et al.-style object channels, so why do we treat objects in experiment 3 like we do in experiment 1?

The motivation for our decision is this. In many practical situations, we cannot make use of the template-based generation of object channels for multiple reasons, three of which we have already included in our Related Works-discussion of the paper. (In brief, these reasons were continually-changing environments, quality of templates, and scalability.) Considering that templates may not always be available, by extension, a background channel—which also is made manually—should arguably also be revoked when transitioning from the 'ideal' approach of Li et al. towards a more practical and generally applicable method as presented in Goel et al. Thus, we retain the inclusion of 'raw' greyscale screens in the observations for experiment 3.

With the overview of the experiments given above, we structure the Methodology as follows. Three subsections, one for each of the three experiments, cover the methodology, except that we include a discussion on the implementation and any relevant technical details at the end. For each experiment subsection, we have the following sub-subsections. First, we briefly repeat the rationale for conducting the experiment, followed by a discussion of the environment and the agent, including the network by which the latter derives its action-values. Although the underlying environment will remain the same—it builds on the OpenAI Gym for Atari 2600 games (Brockman et al., 2016; Bellemare et al., 2013)—the observation generation may vary due to the inclusion of object and motion mask channels. We close the experiment's subsection with details on the precise experimental setup, such as the number of iterations the agent-environment interaction was run for.

Experiment 1

Rationale. The methodology of experiment 1 revolves around two objectives. First is the adaptation of Li et al.

(2017)'s methodology such that it more closely matches what was done in Mnih et al. (2015) and Castro et al. (2018). Second is establishing how we prepare the average return curves and object saliency maps—not just for experiment 1, but for the remaining two experiments as well. As we will see, the method by which we generate OSMs is changed slightly so as to obtain maps that are less susceptible to variation in model configuration. Finally, note that Li et al. compare multiple models to their object-augmented counterparts; we concentrate solely on the Nature DQN.

Environment. Perhaps counterintuitively, we have placed the responsibility of creating object channels on the environment instead of the agent. Our reasoning is that environments provide the observations to which the agent prepares a response in the form of an action. Hence, since object channels are part of observations, we moved their creation to the environment instead of the agent.

The emulated Atari 2600 environments provided by the ALE will form the 'regular' component of our environment, separate from the object-channel component. As we already discussed the ALE in the Related Works, namely in the presentation of the original Nature DQN, here, we focus primarily on the addition of object channels. To be specific, this subsection briefly covers general differences between our environment and that used in Li et al., followed by a discussion on object-channel creation, how this creation differs from Li et al.'s methodology, and why we chose to diverge on such points. We close this subsection by presenting the templates used in this thesis, along with their thresholds and how these were obtained.

General environmental differences. We begin with general environmental differences. Somewhat problematically, Li et al. (2017) do not mention within their paper how they set up their environment, nor do they provide a repository with their code so that we may inspect there the details of their implementation. Without further knowledge, in this thesis, we assume that the authors implemented their environment identically to the one used originally in Mnih et al. (2015); this assumption is suggested to be warranted when inspecting the paper, where it is stated that "[the authors] implemented deep Q-networks (Mnih et al., 2015), double deep Q-networks (van Hasselt, 2010), dueling deep Q-networks (Z. Wang et al., 2016), and advanced actor-critic model [sic] (Mnih et al., 2016) as baselines" (p. 146). With this addressed, the differences in environmental setup can be found in Table 1.

What can be noticed immediately is that the environments differ in three regards.

The first difference revolves around the use of a termination signal on life losses. As we explained in the Related Works, some games in the ALE use lives as a means of allowing the player to try multiple times before a 'game over' is reached; the reason why one may want to regard life losses as 'game over' events, then, is to prevent redundancy in trajectories. We do not follow Mnih and colleagues' approach—episodes are played in full, until all lives are lost—because that way we retain the possibility to compare to the baselines provided by Castro et al. (2018), which in their experiments also deactivated this option. The reason why we do not compare directly to Li and co-authors is that their implementation also differs on other fronts from the original Mnih et al. setup (to be discussed further below) which makes direct comparison difficult regardless of the choice for the terminal on life losses hyper-parameter. We opted for Castro et al. (2018) as a baseline for two reasons: (I) we build on their implementation, and (II) Castro et al. designed their program to be "reliable and reproducible" (Castro et al., 2018, p. 4), which may enable us to compare to their results relatively easily.

The second implementation difference is the maximum number of random NO-OPS to allow upon entering a new episode. Castro et al. effectively disable this feature, and in extension, we do so as well. The same argument from before applies here, too: in order to retain comparability with the baselines, we follow this methodological decision, even though it deviates from what Mnih et al. used.

The NO-OP mechanism at the start of gameplay is designed to inject stochasticity into the environment. The reason for doing so is that Atari 2600 games, if left unmodified, are fully deterministic. In turn, an agent with a sufficient capacity to store representations may simply 'memorise' certain successful trajectories without generalising to light deviations of said trajectories. Deep learning models have this characteristic, and demonstrations in the past have shown that memory capacity may indeed hurt the ability to generalise, for example in the domain of vision (Goodfellow, Shlens, & Szegedy, 2014). We disable the use of NO-OPS at the start of episodes for the same reason we chose not to regard losses of lives as episode terminations.

To make up for this deficiency in environmental stochasticity, we introduce 'sticky actions', as recommended by Machado et al. (2018, pp. 535–538) and as used in Castro et al. (2018)—this is immediately the third implementation difference between our environment and that of Li et al. With sticky actions, at every time step at which the agent may select a new action, there is a $\varsigma \in [0, 1]$ chance of ignoring the agent's action input and instead repeating the previous frame's action input:

$$a_t = \begin{cases} a_{t,\,\text{agent}} & \text{With } 100 \cdot (1 - \varsigma)\,\% \text{ probability,} \\ a_{t-1} & \text{With } 100 \cdot \varsigma\,\% \text{ probability.} \end{cases} \tag{40}$$

Here, we use $a_{t,\,\text{agent}} \in \mathcal{A}$ to signify the agent's intended action at time step $t \in \mathcal{T}$, regardless of whether this action is executed ($a_t = a_{t,\,\text{agent}}$) or not ($a_t \neq a_{t,\,\text{agent}}$, unless $a_{t-1} = a_{t,\,\text{agent}}$).

Machado and colleagues recommend to use sticky actions over NO-OPS for three reasons. First, initial NO-OPS may be relatively ineffective in certain games that have a series of still frames before the game starts. Examples include *Free-*

Table 1
A comparison of Mnih et al. (2013)*'s environmental hyper-parameters with those we used.* Hessel et al. (2018)*,* Li et al. (2017)*, and* Goel et al. (2018) *all used the same hyper-parameters as those used by Mnih and colleagues, with one exception:* Li et al. (2017) *do not use reward clipping. Apart from this, we note that we used the same set of hyper-parameters across all three experiments.*

Hyper-parameter	Mnih et al.	Us	Description
Terminal on life loss	Yes	No	Do life losses end episodes?
Frame skip range	4	4	Number of frames to repeat the same action in.
Max. NO-OPS	30	0	Maximal random number of NO-OPS to use at episode starts.
Reward clipping	Yes	Yes	Whether to apply $\max(-1, \min(1, r))$ to rewards $r \in \mathcal{R}$.
ς	0	0.25	The sticky action probability.

way (where a game type needs to be selected first) and *Ms. Pac-Man* (where an opening jingle plays at the start). Second, and perhaps most obviously, the environment is deterministic beyond the NO-OP number offset in time steps at the start of episodes; this makes memorising trajectories more difficult, but still not as difficult as introducing stochasticity at every decision point, across full gameplay trajectories. Finally, methods that actively exploit the deterministic character of Atari 2600 games (called 'brutes' in Machado et al., 2018) can still work well in environments using NO-OP starts instead of sticky actions. Besides these reasons, the comparability-to-baselines argument applies here as well.

Apart from this, we highlight a subtle but important detail in Table 1: Li et al. (2017) do not use reward clipping—while we, as well as Mnih and colleagues, do—because, Li and colleagues argue, objects become hard to distinguish if reward clipping is active. Put more precisely: objects may be harder to distinguish in terms of action-values if the reward signals that inform these Q-values are all similarly-valued. Specifically, most rewards in Atari 2600 games lie outside the range $[-1, 1]$, such as the pellet-eating scores mentioned in the Related Works section of Mnih et al. for the game *Ms. Pac-Man*. We deliberately choose reward clipping over richer signals for object generation, due, in part, to the usual reason of comparability with Castro et al. (2018). The other primary reason is improved environmental stochasticity, as explained just now.

Aside from this, we will only study two Atari 2600 games in this study: *Pong* and *Ms. Pac-Man*. In contrast, Li et al. (2017) tested on five games: *Freeway*, *Riverraid*, *Space Invaders*, *Bank Heist*, and *Ms. Pac-Man*. Thus, we (I) test on less games, and (II) choose to share one game in common, while testing one other, different one (*Pong*). This raises two questions: "Why do we not run tests for all five games?" and "Why choose a game that cannot be compared with Li et al.'s results?".

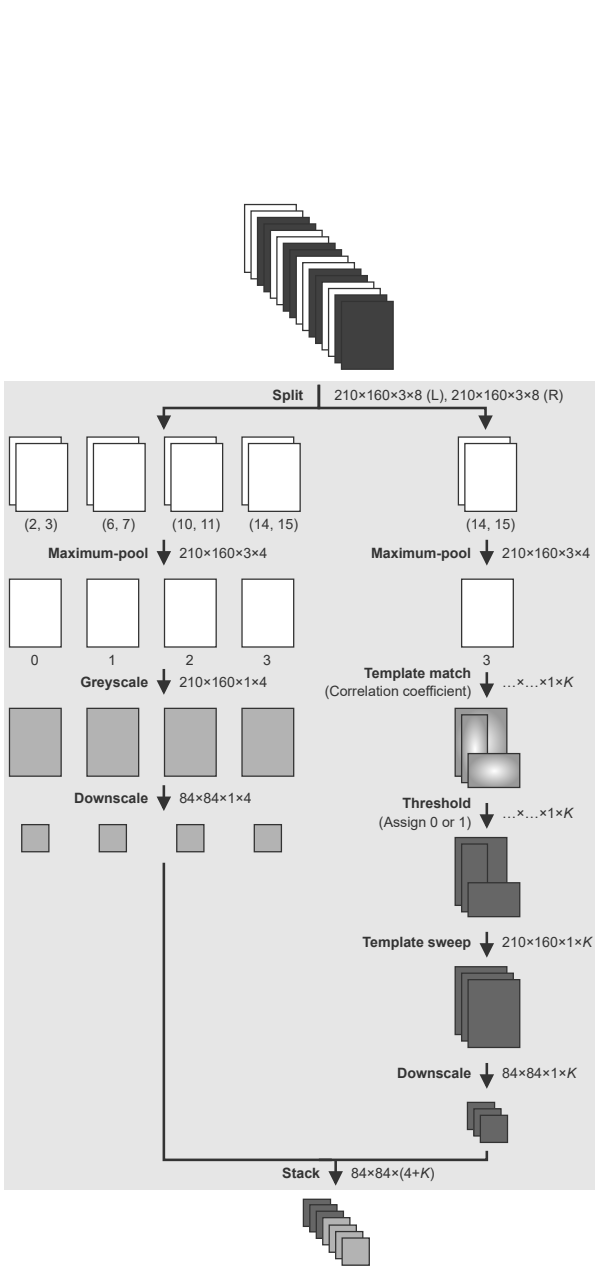
To address the first question, we choose to prioritise assignment of our computational resources to complete all three experiments, and to run multiple repetitions of experiments where possible, instead of obtaining diverse results in only one experiment. Note further that our computational resources are limited. The second question's answer is that we would like to compare experimental results on an 'object-intensive' game (such as *Ms. Pac-Man*) with those of a relatively 'object-deprived' game, such as *Pong*; it may be that the results claimed in Li et al.'s paper do not extend to such games.

Object-channel creation. Having covered the hyper-parameter differences, we now move on to adaptations made to the construction of observations. In order to make it easier to view the changes made at a glance, we provide Figure 2.

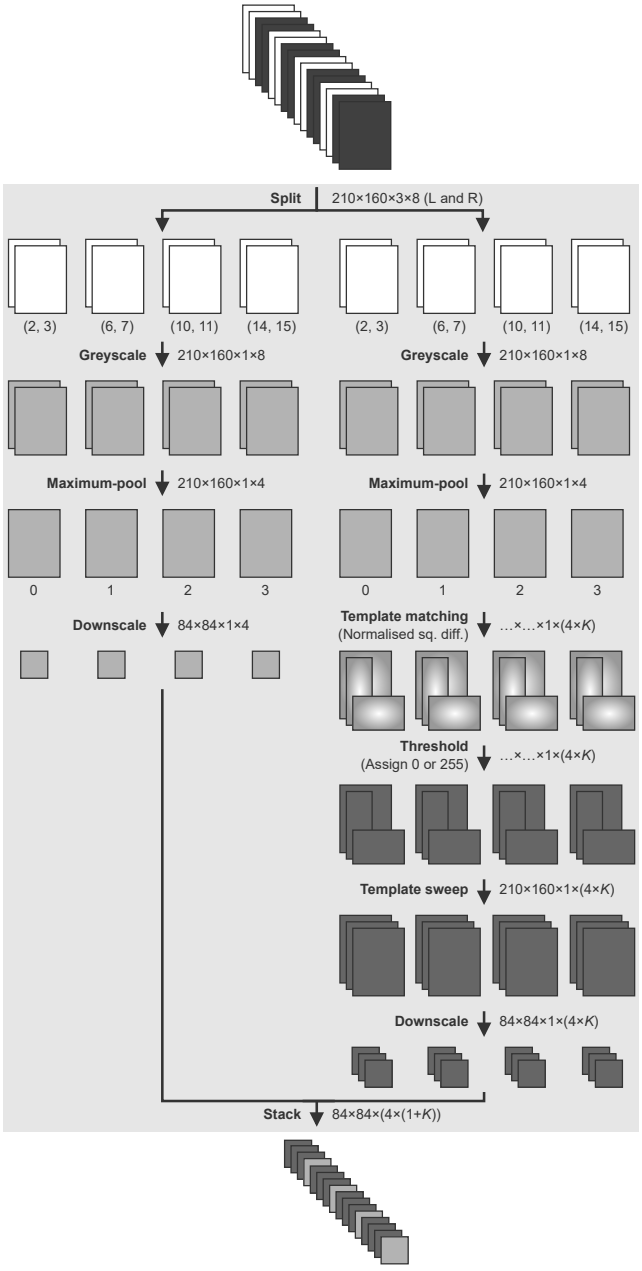
In their paper, Li and colleagues do not describe the procedure by which 'raw' greyscale input frames are processed. As we stated before, we assume that the authors follow Mnih et al. precisely, apart, of course, from the object channel steps. If our assumption holds, then the workflow for producing the 'regular', non-object component of environmental observations is to: (I) take the entry-wise maximum over frames with their direct predecessors, (II) extract the luminance maps from these pooled results, and (III) downscale the result to 84×84 pixels. This is what is shown in the left half of Figure 2a. As can be seen in Figure 2b (left side), we take the same steps in this regard.

Of arguably greater interest is that same Figure's right half, which displays how Li and co-authors compute object channels. Following the authors' description, we begin by max-pooling over the last ('newest') two frames, followed by template matching using $K \geq 1$ objects, which produce K 'matching maps' of variable sizes. Li and colleagues use the correlation coefficient method (discussed in the Related Works; see Equation 30) because of its excellent performance and suitability for practical applications (p. 5). We instead opt for the normalised squared difference (Equation 27) as our experimentation found that it produced better results.

Now, we take a considerable departure from the aforementioned approach by template-matching not over just the last two frames, but over all four frame pairs that eventually form an observation; see Figure 2b (right side). There is a



(a) Li et al. (2017). Note (I) the switch of max-pooling and greyscale conversion, and (II) the use of only the last (newest) pair of frames for the creation of object channels.



(b) Our environment's observation pipeline for experiment 1. Note that, for each max-pooled frame in the stack, we compute a set of K object channels separately.

Figure 2. Observation pipelines for the environments of Li et al. (2017) and the one used by ourselves in experiment 1. Both presented figures show how we transform 16 consecutive 'raw' RGB-coloured screens provided by an Atari 2600 OpenAI Gym environment (Brockman et al., 2016).

Numbers under single frames and number pairs under pairs of frames indicate the frames' indices in the original set of 16 frames: near-black frames are skipped while white frames are used. As is the case in Figure 1, K represents K, the number of objects to create channels for. Furthermore, (L) and (R) refer to the left and right pipeline streams, respectively: left is meant for 'raw' greyscale channels, while the right is meant for object channels. Aside from this, the … in the template matching and thresholding steps point out the fact that, after template matching, we have a per-object variably-sized set of channels, because each object may have a differently-sized template. The phrase '(Assign 0 or 1)' (or 255) means to enter a luminosity value of 0 for below-threshold pixels, and to use 1 or 255 for above-threshold pixels.

clear computational penalty to doing so, but we argue that the benefit outweighs the cost. Specifically, our decision is related to a potential reason[8] why Mnih and colleagues originally proposed to stack four preceding frames to create observations: such stacks may give a deep learning model the necessary historical context to determine whether actions taken in the current state (in the last, newest frame) are appropriate. Think, for instance, of playing *Freeway*: knowing in which directions the cars are moving is crucial in determining whether the current state-action pair should be assigned a relatively high or low action-value. One could object that this historical information is captured in the 'raw' greyscale channels. This is true for experiment 1, but within experiment 2—as we will see below—we drop this source of information. In order to retain historical context there, and to preserve comparability between experiments as much as possible, we argue that computing templates across all four frame pairs is to be preferred.

We pause for a moment to explain why matching maps may be of variable size. Refer back to our discussion on template matching in the Related Works section of O-DRL. Recall, in particular, the definitions of H_{sim} and W_{sim} (Equation 26, the variable qualification under the main equation). In these definitions, we see that as H_{template} or W_{template} increases (or both increase), H_{sim} or W_{sim} decrease, as—intuitively speaking—the template has 'less pixels' to sweep over, thus producing a smaller output map. Vice-versa for decreasing H_{template} and W_{template}. Crucial to observe now is that H_{template} and W_{template} may vary across the objects being detected, thus producing differently-sized maps. We emphasise the potential difference in sizes in the subfigures of Figure 2 by displaying the 'matching maps' with irregular sizes. Of course, it is not strictly required that objects have differently-sized templates; it is just the general, common case.

After template matching, we move on to thresholding. Here, Li et al. state that "[i]n each channel, for the pixels belong[ing] to the detected object, we assign [a] value [of] 1 in the corresponding position, and 0 otherwise" (p. 6). From this statement it is thus not clear when precisely a pixels is deemed to belong to a detected object; this is determined by the threshold, but its value is not mentioned anywhere in the text. Apart from this, we remark that our approach to differentiating between 'object pixels' and 'non-object pixels' is slightly different. In particular, we assign the maximal luminance value of 255 to above-threshold matching map pixels in order to maximally differentiate them from non-object pixels, which we assign a luminance value of zero. In principle, this difference should not be significant, as the model that processes the object channels learns to adjust its weights in accordance to the magnitudes of those channels, but we cannot state this with certainty.

The next step—'template sweeping' as we call it—is identical between Li et al.'s implementation and that of ours. As was the case with the thresholding step, the authors do not mention this step anywhere, but it is required to be able to recover equally-sized object maps that can be used alongside the 'raw' greyscale channels. This step has an inverse effect on the object channels' dimensions compared to what was done in the template matching step, in the sense that all channels recover a uniform size of $H_{\text{image}} \times W_{\text{image}}$ pixels after the computation. In our case, of course, $H_{\text{image}} = 210$ and $W_{\text{image}} = 160$. The algorithm underlying template sweeping is covered in Algorithm 1. While reading, replace M by M_{CC} and M_{SD} for Li et al.'s implementation and that of our own, respectively.

The resultant set of C_k^{object} matrices would correspond to the output of the 'template sweep' step of Figure 2, for both subfigures. The name, as may now have become clear, comes from the fact that we not only assign the top-left positions of matching pixel locations the 'fill-in value', but all positions of the template's 'silhouette' at said position. In this manner, we obtain the block-like contents that can be seen in Iyer et al. (2018), at p. 147.

Finally, the last two steps of the observation pipeline are to: (I) downscale the object channels to 84×84 pixels per channel, and to (II) concatenate these resized channels with the regular resized channels. Our logic is almost identical to Li et al. here, except that we need to concatenate object channels for all four frame pairs. This is the reason behind our different concatenation strategy, appending regular and object channels in an interleaving manner, starting from the 'oldest' pair of frames (with indices 2 and 3) and ending at the 'newest' frame pair (with indices 14 and 15). In contrast, Li et al. simply append their object channels to the end of the regular channels.

Templates and threshold determination. This nearly completes our discussion of the environment. One crucial aspect that we have not made concrete yet is what templates and thresholds we use, precisely. These are shown in Figure 3, in the topmost two rows. Each of the Figure's rows corresponds to a single game for which we define templates, and the columns on that row correspond to single templates, meant for targeting one or more individual objects of the same 'class' in a 'raw' screen observation. For example, the 'Paddle' template is meant to capture both the player's as well as the opponent's paddle in *Pong*. *Fishing Derby* and *Freeway* are included in the Figure as well, even though they will not be discussed within the main text; we have performed some additional experimentation that we present in Appendix E, and include the templates here for completeness.

We have determined the thresholds empirically, although relatively informally, in the following way. For each game, each template's matches map was computed for various states

[8] In neither their 2013 arXiv report (Mnih et al., 2013, p. 5) nor the official Nature publication (Mnih et al., 2015, p. 534) do Mnih and colleagues argue why stacking is used.

Algorithm 1: Template sweeping.

Input : First, a set of $M^{\text{M}}_{\text{sim}}(I_{\text{image}}, I_{\text{template};k}) \in \mathbb{R}^{H_{\text{sim};k} \times W_{\text{sim};k} \times 1}$ template matching result maps, with $k \in [0, K\rangle$ and $K \geq 1$. For each k, $H_{\text{sim};k} \in [1, H_{\text{image}}]$ and $W_{\text{sim};k} \in [1, W_{\text{image}}]$. Second, a set of template matching thresholds $\tau_k \in \mathbb{R}$. Third, a 'fill-in value' $v_{\text{fill}} \in \langle 0, 255]$. For Li et al. (2017) v_{fill} is set to 1 while we set it to 255 instead.

Output : A set of $C^{\text{object}}_k \in \{0, v_{\text{fill}}\}^{H_{\text{image}} \times W_{\text{image}} \times 1}$ not-yet-resized object channels. Again, $k \in [0, K\rangle$ and $K \geq 1$.

begin
  for $k \in [0, K\rangle$ **do**
    $C^{\text{object}}_k \longleftarrow \mathbf{0}$ // Size: $H_{\text{image}} \times W_{\text{image}} \times 1$
    for $\left(y^{\text{object}}_k,\ x^{\text{object}}_k\right) \in \left\{ (y, x) \;\middle|\; M^{M}_{\text{sim}}(I_{\text{image}}, I_{\text{template};k})_{y,x} > \tau_k \right\}$ **do**
      // Find template matching positions with above-threshold similarity scores.
      for $(\Delta y, \Delta x) \in \left\{ (\Delta y, \Delta x) \;\middle|\; \Delta y \in [0, H_{\text{template}}\rangle,\ \Delta x \in [0, W_{\text{template}}\rangle \right\}$ **do**
        // For every position of a template's silhouette, fill said position with v_{fill}.
        $C^{\text{object}}_{k;\, y^{\text{object}}_k + \Delta y,\, x^{\text{object}}_k + \Delta x} \longleftarrow v_{\text{fill}}$
      end
    end
  end
  return $\left\{C^{\text{object}}_k\right\}_{k=0}^{K-1}$
end

of the game. These maps were plotted, and a threshold was sought that would capture only the object of interest across all considered states. By playing the games for multiple hundreds of frames, and by frequently computing and viewing template matches maps, the thresholds in Figure 3 were obtained.

Attentive readers have likely noticed at this point that the templates are in RGB, while our observation pipeline eventually yields greyscale images. We have resolved this discrepancy by converting our templates to greyscale as well, and again empirically testing various thresholds for the templates. The thresholds shown under the templates apply to the greyscale variants of the templates.

Three other remarks need to be made regarding the templates. First is that the last column of each row represents a so-called 'background' template, meant for matching walls or paths. Importantly, these channels are only used in experiment 2, where such environmental information cannot be provided by the 'raw' greyscale input, as it is omitted. In both experiments 1 and 3, background channels are left out from the templates. Second, templates mentioning the keyword 'mask' in their first description line rely on the masking extension to template matching, explained in the Related Works discussion of Li et al. (2017). In particular, any pixel adopting the light-grey background color is a deactivated pixel, due to the mask. Then, third and last, we remark that some templates do not fully resemble their targeted objects. For example, the *Ms. Pac-Man* template Ms. Pac-Man is a simple 3×3 block, while *Pong*'s Paddle is a single, horizontal strip of the player's paddle, with background padding on the left and right. These templates are not erroneous, but have been designed with the specific goal of improved template matching. For example, using only a block for Ms. Pac-Man allows us to both match the player character in the maze, as well as miniature versions of her in the bottom-left corner of the screen, representing lives; it also accounts for various different poses of Ms. Pac-Man.

Agent. In contrast to the environmental differences between our implementation and that of Li et al., the approach to using models is largely identical. Specifically, we use one of the models that is also considered in Li et al.'s procedure, namely the Nature DQN. Furthermore, we adopt the same optimiser.

Still, there are four differences that need to be made clear.

The first is perhaps most immediate: we only use the Nature DQN, while we previously cited that Li et al. also work with the Double DQN (van Hasselt, 2010), the Dueling DQN (Z. Wang et al., 2016), as well as Asynchronous Advantage Actor Critic (A3C; Mnih et al., 2016). We limit ourselves to just the Nature DQN due to the aforementioned arguments on assigning computational resources. Moreover, we are chiefly interested in augmenting the input with objects; testing the impact of such augmentation on multiple games and networks is a concern that is secondary—although of course of importance.

The second difference is a direct consequence of our alternative approach to the environment's observation pipeline. To be precise, the difference in output dimension of these pipelines requires us to reformat the Nature DQN's input dimension to a different shape than what was used by Li and col-

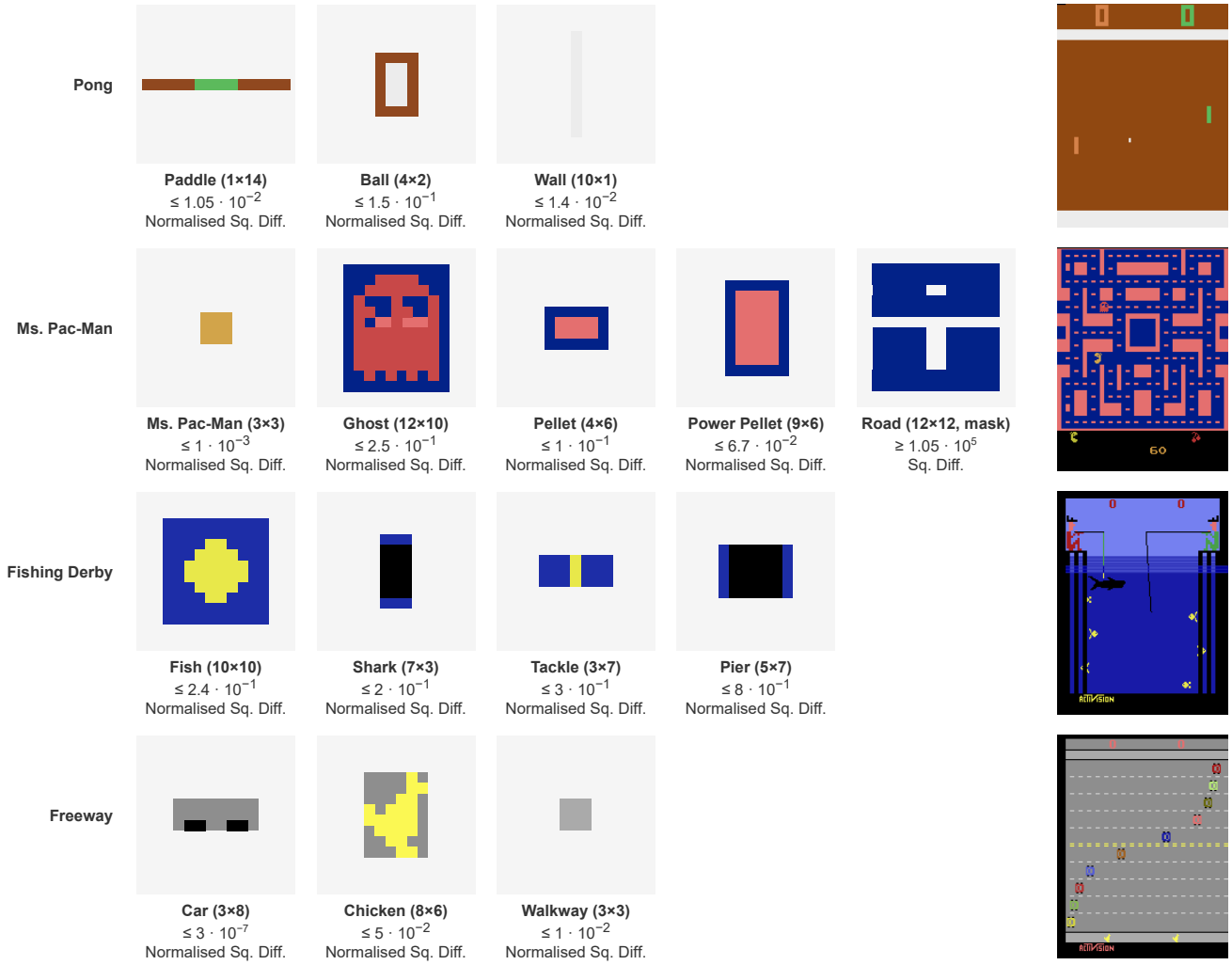


Figure 3. The templates used to detect objects using Li et al. (2017)'s template matching method. Each row shows, for a single game, what objects were targeted using template matching; the last two games are discussed in Appendix E and do not partake in the main discussion; they are included here simply for completeness. Under each template are three lines. The first states the targeted object's name and its $(H_{\text{template}}, W_{\text{template}})$ in pixels. The next line shows the threshold used. Note that all templates use a maximum value for the threshold, except *Ms. Pac-Man*'s Road template, which uses a minimum. The last line shows the template matching technique used; Sq. Diff. stands for 'square difference' (Equation 27), and the Normalised qualifier refers to the normalisation extension to template matching, covered around Equation 31. In the rightmost column, we show example gameplay frames from the respective games for reference. For other details, refer to the discussion of the environment in the Methodology's 'Experiment 1' subsection.

leagues. To see why, compare the output of the environment observation pipelines, shown diagramatically in Figures 2a and 2b: Li and co-authors obtain stacks of size $84\times84\times(4+K)$, while we extract stacks with shape $84 \times 84 \times (4 \times (1 + K))$, caused by our computation of object channels over all input pairs instead of just the last pair. Thus, while Mnih and colleagues originally required an input of $84 \times 84 \times 4$, as all frames were simply 'raw' greyscale screens, Li et al. need an input of the form $84\times84\times(4+K)$ instead. We, in turn, deviate from this mold by working with stacks of the aforementioned $84 \times 84 \times (4 \times (1 + K))$ form.

Third, we use a model loss that is different from the default loss used by Mnih et al. (2015). Originally they used the

Table 2

A comparison between the models relevant to experiments 1 and 2, in terms of hyper-parameters. The only model absent from the comparison is Goel et al. (2018)*'s* MOREL *model, for which we show comparisons in a separate table. Note that* Hessel et al. (2018)*'s model has been tested with* (I) *a greedy policy and noisy layers* (Fortunato et al., 2018)*, and* (II) *with an ε-greedy layer and no noisy layers. In the end, the authors opted for* (I)*, but since we use ε-greedy policies throughout this thesis, we wanted to highlight this fact in this overview. Apart from this, $\varepsilon_{\text{evaluation}}$ represents a constant value, and not a different final value for a linear schedule. The parameters under the 'Us' columns were derived from Castro et al. (2018)'s framework. Specifically, we used their* `dopamine/dopamine/agents/dqn/configs/dqn.gin` *and* `dopamine/dopamine/agents/rainbow/configs/rainbow.gin` *files* (Castro et al., 2022) *for the left and right 'Us' columns, respectively.*

	Nature DQN		*Rainbow*		
Hyper-parameter	**Mnih et al. (2015)**	**Us**	**Hessel et al. (2018)**	**Us**	**Description**
Architecture	As in Fig. 1a	As in Fig. 1a	As in Fig. 1b	As in Fig. 1b	The model's topology and structure.
γ	0.99	0.99	0.99	0.99	The future discount factor.
Policy	ε-greedy	ε-greedy	ε-greedy or greedy	ε-greedy	The policy that the agent uses.
ε schedule	$1 \to 0.1\,(0.05)$	$1 \to 0.01\,(0.001)$	$1 \to 0.01\,(0.001)$	$1 \to 0.01\,(0.001)$	The linear exploration schedule. Notation: $\varepsilon_{\text{start}} \to \varepsilon_{\text{final}}\,(\varepsilon_{\text{evaluation}})$
ε time steps	$1 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	ε annealing time steps.
B_{replay}	$1 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	$1 \cdot 10^6$	The size of the replay buffer.
$t_{\text{learning start}}$	$2 \cdot 10^5$	$8 \cdot 10^4$	$8 \cdot 10^4$	$8 \cdot 10^4$	Time step at which learning starts.
B_{batch}	32	32	32	32	The minibatch size.
m	1	1	3	3	The multi-step update parameter.
θ_t update frequency	16	16	16	16	Online net update freq. (time steps).
$\overline{\theta}_t$ update frequency	$4 \cdot 10^4$	$3.2 \cdot 10^4$	$3.2 \cdot 10^4$	$3.2 \cdot 10^4$	Target net update freq. (time steps).
Optimiser	RMSProp	RMSProp	Adam	Adam	The type of optimiser used.
α	$2.5 \cdot 10^{-4}$	$2.5 \cdot 10^{-4}$	$6.25 \cdot 10^{-5}$	$6.25 \cdot 10^{-5}$	The learning rate.
$\widehat{\varepsilon}$	$1 \cdot 10^{-2}$	$1 \cdot 10^{-5}$	$1.5 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$	Numerical stability scalar for optimiser.
ρ	0.95	0.95	–	–	RMSProp's moving average decay.
Momentum	No	No	–	–	Optional RMSProp momentum.
Centered	Yes	Yes	–	–	Center RMSProp squared gradients?
β_1	–	–	$9 \cdot 10^{-1}$	$9 \cdot 10^{-1}$	First Adam hyper-parameter.
β_2	–	–	$9.99 \cdot 10^{-1}$	$9.99 \cdot 10^{-1}$	Second Adam hyper-parameter.
Type of $\mathcal{B}$	Uniform	Uniform	Prioritised	Prioritised	The experience replay type used.
ω	–	–	$5 \cdot 10^{-1}$	$5 \cdot 10^{-1}$	The priority exponent.
β_{IS} schedule	–	–	$0.4 \to 1$	$0.5\,(\text{constant})$	The priority importance sampling exponent. Notation: as with ε schedule.
β_{IS} time steps	–	–	$2.5 \cdot 10^5$	–	β_{IS} annealing time steps.
N_{atoms}	–	–	51	51	Rainbow's distribution support size.
$[G_{\min}, G_{\max}]$	–	–	$[-10, 10]$	$[-10, 10]$	Rainbow's distribution's support range.
d_{noisy}	–	–	$\mathcal{N}(0, 0.5)$	–	Rainbow noisy layers' distributions.

squared error (SE) to inform updates to network parameters,

$$\mathcal{L}_{\text{SE};\,t}(\theta_t) = \mathbb{E}_{\pi_{\theta_t}}\Big[\Big(R_{T'+1} + \gamma \max_{a \in \mathcal{A}} Q\big(\text{F}(S_{T'+1}), a; \overline{\theta}_t\big) - Q(\text{F}(S_{T'}), A_{T'}; \theta_t)\Big)^2\Big], \tag{41}$$

where T' is a uniform random variable whose support is $[\min(0,\, t - B_{\text{replay}}),\, \min(0,\, t-1)]$ (for more on this, see the experience replay subsection of the Nature DQN section).

Instead, we follow Castro et al. (2018)'s approach and use an adaptation on the SE known as the Huber loss (Huber, 1964). The motivation for choosing this loss over the default SE is that relatively large temporal-difference errors are assigned relatively small values, potentially avoiding numerical instabilities. Using the Huber loss, our loss equation changes to

$$\mathcal{L}_{\text{Huber};\,t}(\theta_t) = \mathbb{E}_{\pi_{\theta_t}}\Big[h\Big(R_{T'+1} + \gamma \max_{a \in \mathcal{A}} Q\big(\text{F}(S_{T'+1}), a; \overline{\theta}_t\big) - Q(\text{F}(S_{T'}), A_{T'}; \theta_t)\Big)\Big], \tag{42}$$

where h is a transformation applied on the temporal-difference error. Using different variable names to avoid collisions with other definitions in this text, this h is defined as (Huber, 1964, p. 75)

$$h(x) \overset{\text{def}}{=} \begin{cases} \frac{1}{2}x^2 & \text{if } |x| < d_{\text{Huber}}, \\ d_{\text{Huber}}|x| - \frac{1}{2}d_{\text{Huber}}^2 & \text{if } |x| \geq d_{\text{Huber}}, \end{cases} \tag{43}$$

with $d_{\text{Huber}} \in \mathbb{R}^+$. One may now realise why the claim of relatively small transformed TD errors applies when using the

Huber loss: beyond discrepancies of size d_{Huber}, h becomes linear instead of remaining quadratic at all values for x, as would be the case for the SE. For our agent, we choose $d_{\text{Huber}} = 1$.

The fourth and last difference has to do with network hyperparameterisation; for an overview of all the changes that are made, see Table 2, and in particular the *Nature DQN* column group. Under it, the 'Mnih et al. (2015)' column displays the hyper-parameters used by the original Nature DQN authors. Due to a lack of further knowledge, we assume Li and co-authors used the same parameter set. The 'Us' column of the column group is derived from the parameters used in Castro et al. (2018), whose framework we use. In essence, the hyper-parameters have been changed so as to allow for an 'apples-to-apples'-comparison with the Rainbow network (Castro et al., 2022), which we will include in experiment 2.

We train the Nature DQN by applying minibatch gradient descent, as discussed in the Related Works, and as presented there in Equation 20. As is also shown in Table 2, the Nature DQN agent works with this model while following an ε-greedy policy with a linearly decaying exploration rate, transitioning from always exploring at time step 0, to exploring only with 1% chance; during evaluation phases—to be discussed in the next subsection—the exploration rate is set one magnitude lower, namely to 1‰ chance. Furthermore, we use a learning rate α of $2.5 \cdot 10^{-4}$, using the RMSProp optimiser (Hinton, Srivastava, & Swersky, 2012) to update the network's parameters; its parameters can be viewed in Table 2 as well. The online network is updated once every four time steps, while the target network is updated once every 32,000 time steps. The discount factor is set to 0.99. Although the procedure here largely follows Mnih et al., and thus also Li et al., the specific parameters used are different, as can be seen in Table 2.

Experiment. The final subsection of our treatment of the methodology for experiment 1 discusses the experiment-level details of what runs to conduct, with how many repetitions, and what outputs to collect from runs. Let us start with answering the first two questions; the third question requires more elaboration.

Runs. By a 'run', within the context of this thesis, we mean a single execution of an agent's learning algorithm within an environment for a defined amount of time steps. Concretely for experiment 1, we only consider the Nature DQN-driven Q-learning agent on the two ALE Atari 2600 environments *Pong* and *Ms. Pac-Man*. In order to define the number of time steps to run this system for, we need to introduce multiple keywords, originally employed by Castro et al. (2018).

We begin by introducing a function $\text{dur} : \mathbb{N} \to \mathbb{N}$ which returns the duration in time steps of a single episode. $\text{dur}(n)$ is only defined for trajectories in episodic environments. Then, let it be defined as

$$\text{dur}(n) \stackrel{\text{def}}{=} T_n - T_{n-1}. \tag{44}$$

Continuing, we will use the words 'time step' and 'episode' as before, except that an episode now may not only end if a terminal state is reached, but also if a given number of time steps $T_{\text{timeout}} \in \mathcal{T} \setminus \{0\}$ have elapsed. Within the theory of MDPs, this would correspond to introducing an extra terminal state $\mathbf{s}_{\text{timeout}} \in \mathcal{S}_{\text{terminal}}$ that has a probability of 1 to transition to after an episode's trajectory would reach the timeout length:

$$\begin{aligned} &t = T_{n-1} + T_{\text{timeout}} - 1 \text{ and } t < T_n \Rightarrow \\ &p(\mathbf{s}_{t+1}, 0 \,|\, \mathbf{s}_t, a) = \begin{cases} 1 & \text{if } \mathbf{s}_{t+1} = \mathbf{s}_{\text{timeout}}, \\ 0 & \text{otherwise}, \end{cases} \\ &\text{for all } a \in \mathcal{A},\ t \in \mathcal{T} \text{ and } n \in \mathbb{N}. \end{aligned} \tag{45}$$

Since $\mathbf{s}_{\text{timeout}}$ is a terminal state, if Equation 45 applies to the environment under consideration, then no trajectory n of duration $\text{dur}(n) > T_{\text{timeout}}$ is allowed to exist. A proof for this claim is given in Appendix B.

We proceed by introducing the notion of a 'phase'. A phase is a sequence of episodes whose summed durations (in time steps) is no less than $T_{\text{phase}} \in \mathcal{T}$ time steps, and where, if we would omit the last episode, the resultant episode sequence's sum of durations would lie strictly before T_{phase}. T_{phase} can then be specified to obtain phases of different possible lengths. Written mathematically, if $\Phi = [n_{\text{start}},\ n_{\text{end}}]$ is our phase, with $n_{\text{start}}, n_{\text{end}} \in \mathbb{N}$ and $n_{\text{start}} < n_{\text{end}}$ forming the indices of our first and last episode, respectively, then

$$\begin{aligned} \sum_{n \in \Phi} \text{dur}(n) &\geq T_{\text{phase}}, \text{ and} \\ \sum_{n \in \Phi \setminus \{n_{\text{end}}\}} \text{dur}(n) &< T_{\text{phase}}. \end{aligned}$$

The last keyword that we introduce is called an 'iteration'. An iteration consists of two phases: a 'training phase' and an 'evaluation[9] phase', having separate time step limits $T_{\text{phase; train}}, T_{\text{phase; evaluation}} \in \mathcal{T}$. The difference between the two is threefold. First, the agent's action-value network is updated during training, and the online and target networks are synchronised after a set period of time (refer for this to the update frequencies in Table 2); both of these steps are not performed during evaluation. Second, experience replay quadruples $(\text{F}(\mathbf{s}_{t'}),\ a_{t'},\ r_{t'+1},\ \text{F}(\mathbf{s}_{t'+1})) \in \mathbb{R}^F \times \mathcal{A} \times \mathcal{R} \times \mathbb{R}^F$ are only stored during the training phase. Third and last, dynamically-adapted exploration schedules are used solely during the training as well; the evaluation phase uses constant, relatively low exploration values instead.

With the definitions of 'phase' and 'iteration' covered, we are in the position to describe how many runs to perform, and how many time steps these runs should take. For experiment 1, we perform a single run of 60 iterations: one for *Pong*, and another for *Ms. Pac-Man*. For each, we train a separate

[9] Within the literature, one may also encounter the words 'validation' or 'development' instead of 'evaluation'.

network. We specify that training phases must minimally cover $T_{\text{phase; 1; training}} = 1{,}000{,}000$ time steps, while evaluation phases are required to cover $T_{\text{phase; 1; evaluation}} = 500{,}000$ time steps. Further, single episodes are maximally allowed to proceed for $T_{\text{timeout; 1}} = 108{,}000$ time steps. Thus, we have the following experimental configurations grid, where cells display the number of repetitions (in runs) of the configuration it represents:

		Game	
		Pong	*Ms. Pac-Man*
Model	Nature DQN	1×	1×

In order to compare our results, we remark on the following. Mnih et al. (2015) trained their Nature DQN agents for 50 million time steps (p. 534). After that, they recorded undiscounted returns ('scores') for 30 episodes, each lasting maximally 5 minutes, or, put differently, 5 min × 60 sec × 60 Hz = 18,000 time steps maximally. In contrast, Li et al. (2017) used only 10 million frames and recorded scores over 50 episodes (p. 9), with no qualification on the number of minutes maximally played, making it probable that episodes were run until completion. Provided with Castro et al. (2018)'s baseline of the Nature DQN, we can compare our results fairly with those of Mnih and colleagues, whereas we need to take care with Li et al.'s results, as we run for a number of time steps instead of for a set amount of episodes.

Object saliency maps. Arguably central to this thesis is the object saliency map (OSM), as introduced by Li et al. (2017). Since the procedure for creating OSMs may be seen as involved, we have included Figure 4 to give a visual overview besides the explanation of the OSM creation procedure that will now follow. At the end, we discuss how our approach attempts to improve on Li et al.'s proposed method.

The creation of an object saliency map begins with finding a set of consecutive gameplay frames. We immediately need to be careful in order to handle an important detail properly: just as in Mnih et al. (2015), we use both frame skipping and maximum-pooling over frames, and as such we cannot take immediately successive frames. If we would, it could cause out-of-distribution problems when feeding the processed frames into the DRL network. As such, when collecting gameplay frames, we collected the latest six pairs of frames from a 'skip cycle'. By a skip cycle, we simply mean the last two frames from a four-frame block over which is frame-skipped. We need the last two frames instead of only the very last frame because of the max-pooling operation. Furthermore, in contrast to our description of the environment's observation pipeline (Figure 2), we do not start with RGB frames but with greyscale counterparts; the size of the unprocessed frames does remain the same: 210×160 pixels. The scenes we used across all three experiments can be found in Appendix C.

Subsequently, we manually find our pre-specified number

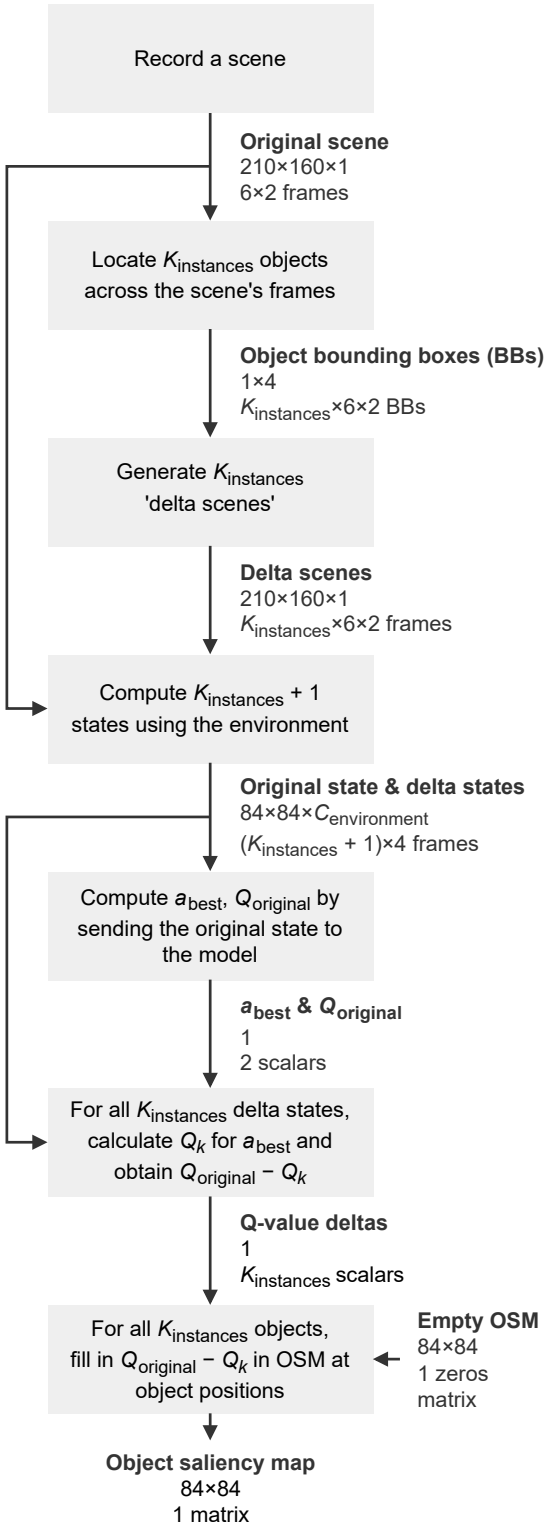


Figure 4. Creation of a single object saliency map (OSM). $K_{\text{instances}}$, $C_{\text{environment}}$, a_{best}, Q_{original}, and Q_k correspond to $K_{\text{instances}}$, $C_{\text{environment}}$, a_{best}, Q_{original}, and Q_k within the text, respectively.

K of objects in the scene. Note that, frequently, either a searched-for object is not present in the frame or it appears within the frame multiple times. An example of the former may be a ball in *Pong* that has just gone off-screen; for the latter case, think, for instance, of *Ms. Pac-Man* pellets. This is why we introduce the symbol $K_{\text{instances}} \in \mathbb{N}$: it represents the total number of objects that has actually been found across at least one of the six skip-cycle pairs. If an object appears in a strict subset of the pairs, then in any frame in which it is absent, its location is assigned to a background pixel. At the end of the localisation stage, we obtain for each detected object ($K_{\text{instances}}$), across all skip-cycle frame pairs (6×2) a bounding box.

Next, we create $K_{\text{instances}}$ 'delta scenes'. A delta scene for an object $k \in K_{\text{instances}}$ is like the original scene from the first step, except that across all skip-cycle frame pairs the object enclosed in the bounding box has been removed. More precisely put: the pixels within the bounding box that encloses k is replaced by the background pixel color. Note that Atari 2600 graphics do not display rich textures, and as such, often a single color can be used to replace an object by background without harming the 'graphical integrity' of the resultant image.

We may now combine the results of the first and third steps of our procedure to compute $K_{\text{instances}} + 1$ states. A state is different from a scene in that it can be directly passed to a DRL model: its shape is $84 \times 84 \times C_{\text{environment}}$, where $C_{\text{environment}} \in \mathbb{N} \setminus \{0\}$. For instance, in experiment 1, $C_{\text{environment; 1}} = 1 + K$ as we combine a 'raw' greyscale screen with object channels. Primarily to avoid manual labour, we send the original and delta scenes to the environment and let it compute the states; see also Figure 2b. At this point, one may wonder why we have supplied six skip-cycle pairs, as the pipeline only requires four. The reason is consistency, as in our third experiment, due to an implementation detail, we require two extra preceding skip-cycles. Strictly speaking for experiment 1, they are not needed.

Once we have the states, we can compute Q-values for them. This is significant, as differences in Q-values—'Q-value deltas' as we call them—form the basis for the OSM. To start in this direction, we compute $a_{\text{best}} \in \mathcal{A}$ and $Q_{\text{original}} \in \mathbb{R}$. The former is obtained by finding the Q-value-maximising action when feeding the original state into the DRL model; the associated Q-value is Q_{original}.

Now, we find all $Q_k \in \mathbb{R}$, with $k \in K_{\text{instances}}$, by sending the $K_{\text{instances}}$ delta states to the DRL model as well, except that we always collect the Q-value at a_{best} instead of maximising over the actions again. Why this is necessary becomes clear when one imagines a situation where, when some object of interest is replaced by background, the Q-valuations per each action change so drastically that a different action becomes better, possibly with an even greater Q-value than the original state. If we would not work with a_{best} here, we would capture the Q-value for taking this different course of action, while we want to evaluate how much 'worse off' we get when the situation changes. Given all Q_k, we compute the Q-value deltas $Q_{\text{original}} - Q_k$. The order is important here, as we want to quantify whether k's *removal* is beneficial or not; swapping the terms would quantify change in (estimated) return when *adding* k to the scene.

Then, the final step in the OSM creation pipeline is relatively simple. First, we create an empty object saliency map $M_{\text{saliency}} \in \mathbf{0}^{84\times84}$, and then fill this map by determining where objects were in the last frame of the input scene, and entering at those pixel locations the Q-value delta for k. An interesting problem here is that order matters: if we have two objects $k_1, k_2 \in K_{\text{instances}}$, which should go first if the two have (partially) overlapping bounding boxes? Within this thesis, we have found that such overlap was rare, and that if it occurred, the overlap was only partial. In those cases, we performed a manual re-ordering of the objects such that both would be partially visible in the resultant map.

Up until now, the procedure we described has closely matched Li et al.'s method (pp. 6–7). Previously, though, we claimed that our approach seeks to improve in multiple respects on the original workflow of those authors. We have three proposed improvements, which we will explain in turn.

First, notice that an OSM is essentially a highly discrete heatmap. A shared problem in visual presentation among all heatmaps is that we have to choose a minimum and maximum value for the heatmap's color range. Normally, this need not be a considerable issue, as we can, for example, pick the minimum and maximum values observed in the heatmap's grid. However, this method is inappropriate if the mapped set of points contains outliers. In such cases, outliers dominate the color range's minimum or maximum (or both), and lead to a drift in color of all values displayed. This problem is exacerbated when the color range is of the diverging type—which means: having a 'low-middle-high' tripartition of color that smoothly interpolates between one another—which would cause most values to be sent to the 'middle' value. Since this middle value is often a neutral color, suggesting neutrality, outliers can seriously affect interpretation of heatmaps. Because OSMs also use a diverging color palette for heatmaps, and as outliers do occur, we need to address this problem.

Our proposal is to collect, across scenes, all Q-value deltas, keeping the experiment's DRL model and game on which said model was tested constant during the collection. (Of course, separate collections can be conducted for other model-game pairs.) Subsequently, we sort the action-value deltas and compute the 10^{th} and 90^{th} percentiles of those action-value deltas, resulting in capturing around 80% of the data across the scenes within the color range; all value deltas outside the range are regarded as outliers. With this approach, across all scene object saliency maps, the presentation is influenced less by outliers. In turn, it may help viewers interpret the OSMs, as

(I) there is a clearer distinction between common action-value deltas (colored within the color range) and notable, irregular action-value deltas (colored at the minimum or maximum), and (II) it facilitates comparison across scenes—keeping the model and video game constant—as all such scenes share an identical color range. We provide boxplots of the Q-value deltas for each experiment–DRL model–video game combination; see Appendix D.

Our second suggested improvement is closely related to the first: if we have values near the minimum or maximum color value, we would arguably like to know whether they are indeed outliers or simply close to, but not surpassing, the color range bounds. To do so, we place a border around each pixel belonging to an outlier; pixels that are not outliers are displayed as-is, without a border. This adaptation further improves differentiation between regular and irregular action-value deltas.

The third and final adjustment is applied only during the second experiment, but we mention it here for completeness. If we perform multiple repetitions of training a DRL model on a game, then all resultant separate models each may produce different OSMs for the same scene. Thus, it is possible to 'average' across all OSMs. The process of taking the average of multiple OSMs works as follows. For each object instance in the considered scene, compute the average of all Q-value deltas for said object instance given by all OSMs we have. Then, create a new, empty object saliency map, locate all object instances, and enter the average Q-value deltas in the corresponding object instance locations. Finally, aggregate all Q-value deltas across all OSMs and compute the 10^{th} and 90^{th} percentile of this 'grand total' dataset of Q-value deltas. Then, use these bounds as the bounds for the average OSM. The motivation behind using average OSMs is relatively simple: it reduces the variance in Q-value deltas caused by the specific parameterisation of individual networks, and attempts to find, per object instance, a relatively model-agnostic delta.

Model output. Now that we have introduced OSMs, we can finally describe what the experiment's intended output is. Concretely, we collect two results from each trained Nature DQN network, trained on *Pong* and *Ms. Pac-Man*: (I) a curve displaying per iteration the episode-averaged, undiscounted return of said iteration's evaluation phase, and (II) per each scene, the model's OSM for said scene.

Experiment 2

Rationale. With the combination of both 'raw' greyscale screen input and Li et al. (2017)-style object channels, we encounter an issue in the interpretation of results—especially the average undiscounted return curves. The reason for this is straightforward, although no less problematic: if our results are different from the baselines provided by Castro et al. (2018), then we cannot, with certainty, attribute this effect to the object channels; it might as well be the case that the greyscale channels remain the only useful channel of information, but this source of information is leveraged relatively effectively on our system due to an external effect, such as using a different version of a machine learning library.

To rule out this possibility, we continue with experiment 2, in which we only consider Li et al. (2017)-style object channels. Simultaneously, we allocate a relatively large portion of our computational resources to this experiment in order to (I) run, besides the Nature DQN, also with Rainbow-driven agents (Hessel et al., 2018), and (II) repeat each configuration 3 times.

Environment. Regarding the environmental setup, most of the details from experiment 1 carry over to experiment 2. The primary difference is the deprecation of 'raw' greyscale screen input; only object channels remain. One caveat prevents us from directly proceeding to the next experimental aspect, however: certain environments store important 'non-object' information in their greyscale inputs which would be lost if we only included object channels in inputs. One example that we already mentioned at the beginning of the Methodology is the set of walls in *Ms. Pac-Man*: it is imaginable that the agent would require a significant amount of time to navigate the maze if it would 'play in the dark'.

To resolve this issue, we introduce a single extra channel to the existing sequence of object channels: the so-called background channel. The background channel is constructed in much the same way as regular object channels, except that the targeted 'object' now is not an object, but rather a structural element. As can be seen in Figure 3, for *Pong* we match the top and bottom walls, while for *Ms. Pac-Man* we match the floor. Since the latter lies behind the objects of interest—mostly pellets—we use a mask to avoid the most frequently-occurring two object classes: pellets and power pellets. Due to an implementation detail relating to OpenCV (Bradski, 2000), we use regular square difference instead of normalised square difference for such masked templates.

The revised observation pipeline can be viewed diagrammatically in Figure 5.

Agent. For experiment 2, we introduce another agent type besides the one driven by the Nature DQN, namely the Rainbow agent (Hessel et al., 2018). Although similar in interface to the Nature DQN—it also requires states as input and returns, for each action, an action-value—Rainbow is known to perform better on most games of the ALE Atari 2600 game suite (Hessel et al., 2018). For our experiment specifically, using Rainbow besides the Nature DQN has an important reason. The Nature DQN may have limited performance in select games, simply because the network's design inhibits further performance increments on the task. In these cases, any potential benefit of object channels may be obscured by the 'inherent limitation' of Nature DQN agents. Hence, using the relatively powerful Rainbow besides the Nature DQN can help

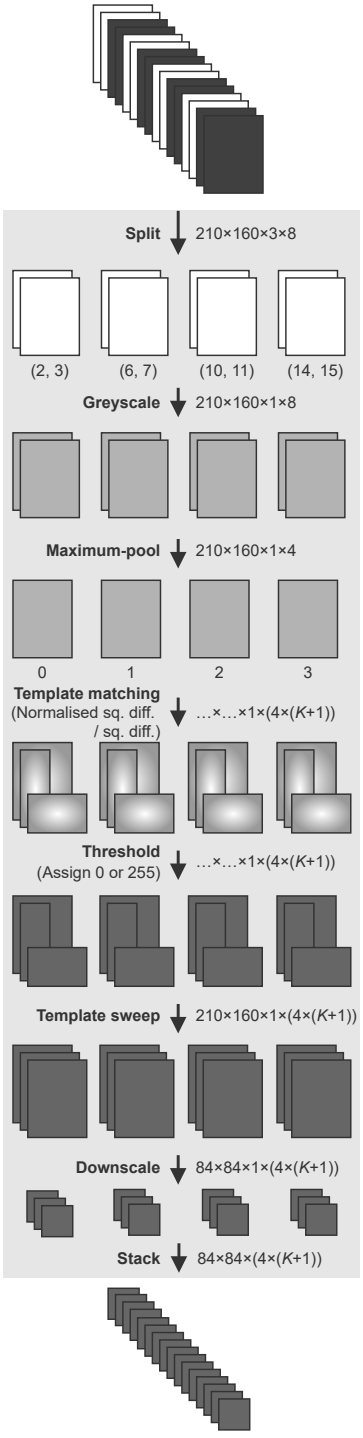


Figure 5. Experiment 2's observation pipeline—a revision to our observation pipeline for experiment 1 (Figure 2b). We now only use Li et al. (2017)-style object channels, plus a background channel; this is the $K + 1^{\text{th}}$ channel. Note that, although we use the normalised square difference for most templates, *Ms. Pac-Man*'s background uses a mask which required us to use the regular, non-normalised square difference. See also the commentary under Figure 3.

us detect these situations, helping us better determine whether object channels are beneficial to agent performance.

The configuration of the Rainbow agent is shown in Table 2, in the second column group. Note that we deviate from Hessel et al. (2018) in that we do not use noisified dense layers. Instead, we keep using an ε-greedy policy, which was already considered as an alternative to noisy layers in Hessel and colleagues' original paper, although they finally opted for noisy layers due to its increased performance (p. 3,220). Our reason to deviate from this choice is primarily to remain consistent with Castro et al. (2018)'s baselines; another argument is that the remaining five extensions to the original Nature DQN are still sufficient to avoid the effect of a Nature DQN's limited performance on detecting object channel benefits.

Besides this, we remark that, as the environment's observation pipeline has changed—it now produces observations of the shape $84 \times 84 \times (4 \times (K + 1))$—the input channels for our Nature DQN and Rainbow agent are changed accordingly.

Experiment. As is the case with the preceding subsections, our overall experimental setup remains largely identical to that used in experiment 1. We make two modifications to the original scheme, to be precise.

First, we run each DRL model-video game combination 3 times instead of just once. Note that the experimental grid has also changed due to our inclusion of Rainbow. Thus, we have the following experimental grid, identical in interpretation to the one we presented in experiment 1:

		Game	
		Pong	*Ms. Pac-Man*
Model	Nature DQN	3×	3×
	Rainbow	3×	3×

Second, notice that we obtain three individual sets of average undiscounted return curve-and-OSM pairs. While in experiment 1 we could use the former directly, we now aggregate the three curves by averaging per each iteration all three episode-averaged undiscounted evaluation phase returns into an 'average-of-averages'. Further, we compute the standard error over the three values in order to communicate spread. Similarly, we aggregate the OSMs by computing their average, as is described in the 'Experiment' sub-subsection of the 'Experiment 1' subsection. Note that we do not consider the background channel while computing the maps, as we do not regard its content as belonging to any object.

Experiment 3

Rationale. Our third and last experiment revolves around using an alternative to the template-based object channels proposed by Li et al. (2017), namely object segmentation masks, introduced by Goel et al. (2018). The chief motivation for working with these channels is that there is an inherent limitation in Li and colleagues' approach: it requires

manually-constructed templates of the environment. Ideally, we would like to obtain agents that can extract the relevant information from the environment fully on their own, as was the case in the original Nature DQN, for instance. Unlike that system, though, we require this information to be less granular and more object-like.

This observation is also made by Goel et al. (2018). Hence, they propose using Motion-Oriented Reinforcement Learning (MOREL) which can detect motion from observations by minimising a reconstruction loss (Equation 36), derived from the principle of optical flow.

Within this experiment, we use the unsupervised component of MOREL to build a counterpart to experiment 1 which does not rely on manually-constructed templates. Once trained, we can compare the systems obtained here with those in experiments 1 and 2 to see whether trading in some of the accuracy gained by using template-based objects for less manual labour still produces any object-based improvement.

Environment

As is the case with template-based object channel creation, we view the task of generating motion-based object channels as belonging to the environment. One may object here that in Goel et al. (2018)'s original design, the unsupervised motion model is part of the overall DRL model, and thus part of the agent. We counter this by pointing out that we only use the unsupervised network component, which can—and is—trained standalone, without any relation to the agent (pp. 5,686–87).

Before continuing, we need to address a terminological detail that has remained unspecified up until now, but which may cause confusion if we leave it that way. Specifically, we have used two terms interchangeably: 'motion mask channels' or 'motion-based channels' on the one hand, and 'object segmentation masks' on the other. Both refer to the same idea: extracting the contents of the 'Object (Segmentation) Masks' block in MOREL's unsupervised submodel, which can be found in Figure 1d's left grey block, just left and below of the 'Optical Flow' block. During the Related Works, it has also been denoted mathematically as $P_k \in [0, 1]^{H_{\text{images}} \times W_{\text{images}}}$, with $k \in [0, K\rangle$. Note that, although $K \geq 1$ was already used to denote the number of Li et al. (2017)-style object channels, we re-use it here for MOREL object segmentation masks. Since both populate the object-based channels, sharing the symbol K, we argue, is appropriate. Note, besides this, that for MOREL, K is a hyper-parameter that determines how many unique objects can maximally (inclusively) be segmented from the input observation. We fix K to 4 within this thesis, as it matches the number of object channels maximally used in the Li et al. (2017)-style object channels from experiments 1 and 2, not considering background channels.

We cannot use the MOREL subnetwork immediately; it needs to be trained in a self-supervised manner first, as is also described in the Related Works section on MOREL. We follow Goel et al.'s procedure and create a single, pre-trained subnetwork for each of the games *Pong* and *Ms. Pac-Man*. This is done in three steps.

First, we let a random policy play the game under consideration until precisely 100,000 processed frames have been collected. These are frames that are not direct successors of one another, but are instead frame-skipped, max-pooled, and subsequently resized to 84×84 pixels, as is shown in the topmost two steps of our observation pipelines for experiments 1 and 2 (Figures 2b and 5, respectively). Then, we aggregate these frames per episode, producing sets $F'_1, F'_2, \ldots, F'_{N_{\text{pretrain}}}$, where $N_{\text{pretrain}} \geq 1$ is the number of episodes that the random policy ran for. Each F'_i, with $i \in [1, N_{\text{pretrain}}]$ stores at least one processed frame.

Second, we sample 250,000 pairs of successive, processed frames by repeatedly performing the following: (I) Select an episode frame set $F' \in \{F'_1, \ldots, F'_{N_{\text{pretrain}}}\}$ proportional to the number of processed frames it has. That is, each set F'_i with $i \in [1, N_{\text{pretrain}}]$ has a selection probability of

$$p_{F'_i} = \frac{\left|F'_i\right|}{\sum_{j \in [1, N_{\text{pretrain}}]} \left|F'_j\right|}.$$

This approach partially avoids training the MOREL subnetwork on only relatively 'underperformant' trajectories, by preferring trajectories in which the agent managed to play for a relatively long number of time steps. (II) Then, draw two consecutive processed frames by uniformly (pseudo-)randomly sampling an index $i' \in [0, |F'| - 1\rangle$ and using $(i', i' + 1)$.

Third, the resultant set of pairs of processed frames are sequentially input to the MOREL subnetwork. At each output, we adapt the network's parameters using the loss described in Equation 38. As is done by Goel and colleagues, we use the Adam optimiser (Kingma & Ba, 2015) with the following hyper-parameters (the symbols having the same interpretation as in Table 2):

Parameter	Value
α	$1 \cdot 10^{-4}$
β_1	$9 \cdot 10^{-1}$
β_2	$9.99 \cdot 10^{-1}$
$\widehat{\varepsilon}$	$1 \cdot 10^{-8}$

Further, we use the same linearly-increasing λ_{reg} as was mentioned in the Related Works section on MOREL.

Once we have a trained model, we can use it to generate object segmentation mask frames for our environmental observations. Recall from the introduction to the Methodology that—unlike experiment 2—we also need greyscale frames to account for the static component of the observations. All in all, the pipeline is shown in Figure 6. In it, the use of six skip cycles at the input is due to our aforementioned implementation detail.

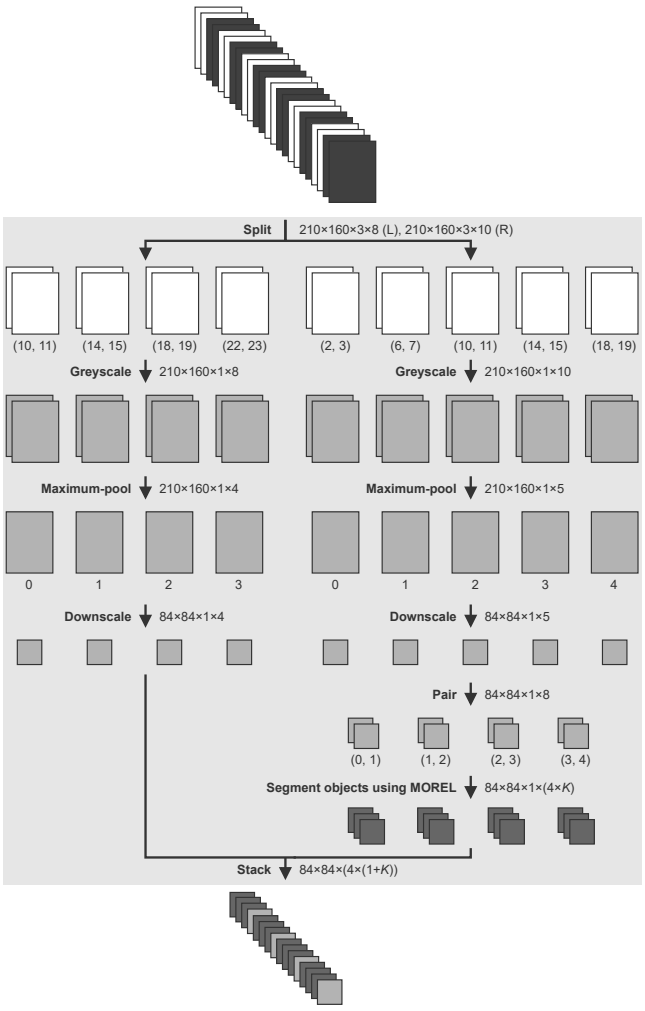


Figure 6. The environmental observation pipeline for experiment 3. It is interpreted similarly as the diagram for experiment 1, so refer to Figure 2b as well.

Agent

We use the same two agents as in experiment 2: the Nature DQN and Rainbow; their parameterisations also remain as-is to facilitate comparison across experiments. Our inputs remain of the same size, too: $84 \times 84 \times (4 \times (1 + K))$. Here we change $K + 1$ to $1 + K$ to highlight the facts that (I) we re-introduce the greyscale frame, which we put first in the sequence of observation channels, and (II) we remove the background channel, which comes last in the channels sequence.

Experiment

For experiment 3, the experimental setup is nearly identical to the one used in experiment 1, except that we now use both the Nature DQN as well as Rainbow. Thus, our experimental configurations grid changes to the following:

		Game	
		Pong	*Ms. Pac-Man*
Model	Nature DQN	1×	1×
	Rainbow	1×	1×

Although we would have been interested in repeating each configuration three times, like we did in experiment 2, we were limited in computational resources and had to prioritise.

Technical implementation

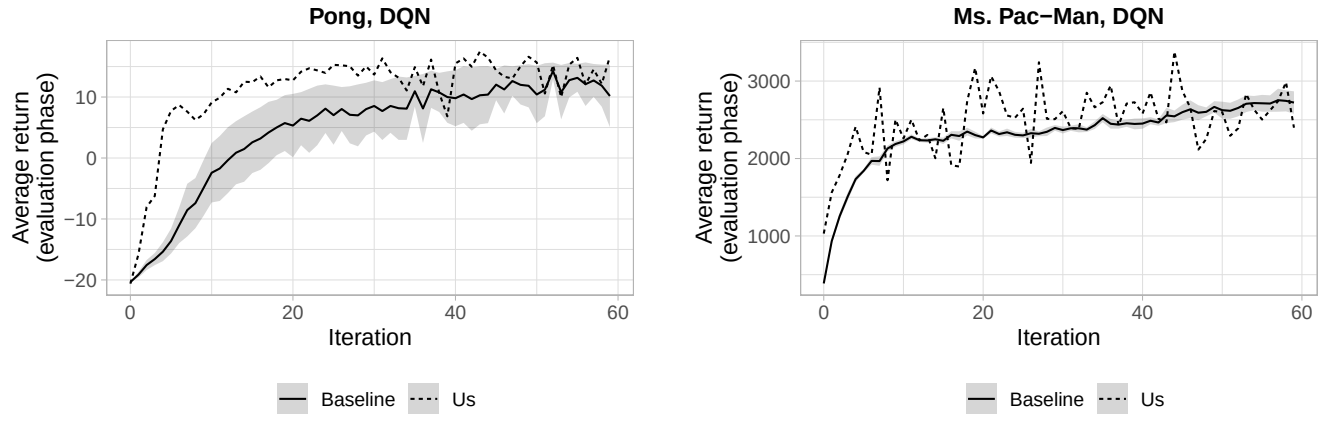
We close the Methodology by referring to software that has enabled the concrete implementation of this thesis' ideas. The overall project has been written in Python (van Rossum, 1995), and is a fork from Castro et al. (2018)'s Dopamine framework. The machine learning framework on which we build is TensorFlow (Abadi et al., 2016). Observation preprocessing is achieved via a combination of OpenCV (Bradski, 2000, primarily for resizing and template matching) and Pillow (Lundh, 1995, Clark & Contributors, 2010). NumPy (Harris et al., 2020) helped with general array programming.

Results

We discuss our results with the same approach that we took with the Methodology: we consider experiments 1, 2, and 3 in turn. For each, we first show the average undiscounted return curves per each DRL model-game combination, followed by a presentation and explanation of interpretation of the object saliency maps. We postpone closer examination of the OSMS until the end of the Results, as then the maps from all three experiments can be compared with one another. Furthermore, experiment 3 has an extra subsection because we also present examples of object segmentation masks there.

Experiment 1

Average undiscounted return curves. We begin our overview of the results by considering the episode-averaged undiscounted evaluation phase return curves for experiment 1; they are shown in Figure 7. While observing the plots, recall that experiment 1's configuration grid does not include Rainbow—it is included from experiment 2 onward. Furthermore, we remark that even though we use reward clipping throughout all experiments, the total undiscounted reward that can be collected varies considerably between the two games. Two reasons for this phenomenon are that episode lengths differ between games, and that rewards may occur more or less frequently depending on the game played. For instance, in *Pong* one gains or loses points relatively infrequently, namely when winning or losing matches. In contrast, in *Ms. Pac-Man*, each single pellet—abundant across the playing field—contributes to the score when eaten.



(a) Nature DQN–*Pong*.

(b) Nature DQN–*Ms. Pac-Man*.

Figure 7. Episode-averaged undiscounted return curves for experiment 1, varying over the two games *Pong* and *Ms. Pac-Man*, but keeping the DRL model—the Nature DQN—the same. The baseline's curve is actually a 'grand average' of individual episode-averaged undiscounted returns at that iteration; the shaded area around it is the standard error around the averages. The baseline used five averages to compute a grand average and standard error at an iteration.

When examining the left subplot (Figure 7a), it appears that the Nature DQN agent reaches similar levels of episode-averaged undiscounted return as does Castro et al. (2018)'s baseline, which uses the 'raw' greyscale screen input without any object channels. However, our object channel-augmented agent seems to reach such levels relatively quickly, speaking in terms of iterations. In particular, if we consider an average undiscounted return of 15 as the 'convergence score', then we could argue that the object channel-augmented Nature DQN reaches this score around iteration 30, while the 'regular', baseline Nature DQN arrives at such a score only past 60 iterations. Indeed, if we view Castro and colleagues' baselines in full—which go on for 198 iterations—it seems that the regular Nature DQN only reached an average undiscounted return of 15 near 120 iterations (Castro et al., 2022).

Moving on to *Ms. Pac-Man*, to the right, in Figure 7b, the situation seems different. Specifically, the object channel-augmented Nature DQN agent appears to yield similar episode-averaged undiscounted returns over the iterations to Catro et al.'s regular, baseline Nature DQN agent. If ran for even longer, it may even have become the case that the object channel-augmented agent stagnated, while the baseline continued to improve.

Taking stock of both graphs, we bring attention to one remaining detail that appears to be shared among both *Pong*'s object channel-augmented agent and that of *Ms. Pac-Man*: the two curves seem more noisy than the baselines, where especially our *Ms. Pac-Man* Nature DQN agent oscillates more in average undiscounted return than the baseline: the standard error around the baseline's curve is difficult to notice, suggesting almost no fluctuation around the 'grand average' of average undiscounted returns across the five runs of which it is composed. Contrast this to the variation in average undiscounted return of our agent, which sometimes varies with almost 1,000 score points—a relatively large amount.

Object saliency maps. Besides average undiscounted return curves, we present object saliency maps of the Nature DQN agent applied to both *Pong* and *Ms. Pac-Man*. The results can be seen in Figure 8, although for understanding the plots we need to explain the following three points.

First, as we have already explained in the Methodology, OSMs display 'silhouettes' of objects. Recall that these have been produced during the template-sweeping step, shown in Figure 5). Consequently, OSMs display rectangles instead of more recognisable object outlines, which may make it difficult to determine which object is present where in the Figure. To address this issue, we provide Table 3 for assistance. As the scenes stay the same across experiments, Table 3 can be used throughout the rest of the Results to find the rectangle belonging to the object of interest.

Second, we need to make precise what the interpretation is of each single colored rectangle in our OSMs. Denote by $\mathbf{s}_{\text{present};k}, \mathbf{s}_{\text{absent};k} \in \mathcal{S}$ the state under consideration with $k \in [0, K_{\text{instances}}\rangle$ present on all channels, and that same state with k removed from all channels, following the procedure outlined in the Methodology, respectively. Then, the value $\Delta_k \in \mathbb{R}$ represented by the color of k's rectangle in the OSM is the action-value difference, evaluated at the Nature DQN, between

1. The action-value of the scene's original state $\mathbf{s}_{\text{present};k}$, choosing the action $a_{\text{best}} \in \mathcal{A}$ that maximises action-value for said state, and

2. The action-value of the delta scene's state, $\mathbf{s}_{\text{absent};k}$,

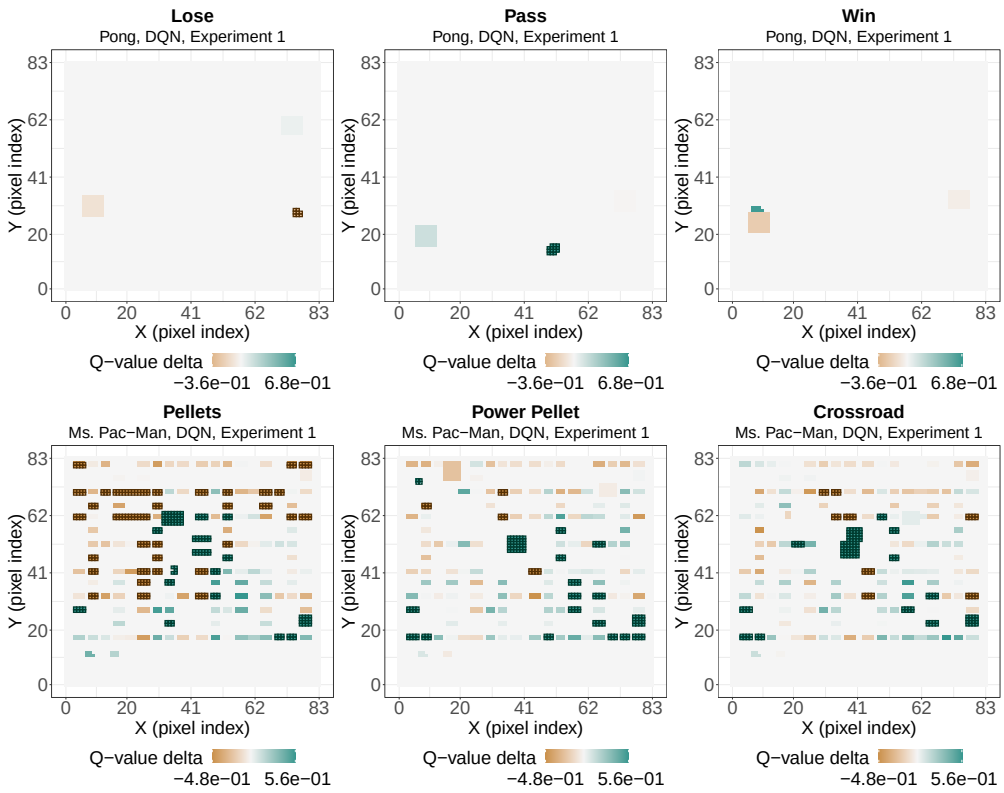


Figure 8. Object saliency maps for experiment 1.

choosing a_{best} again.

Mathematically, this is equivalent to

$$\Delta_k = Q\Big(\text{F}\big(\mathbf{s}_{\text{present};\,k}\big), a_{\text{best}}; \theta_{\text{final}}\Big) - Q(\text{F}(\mathbf{s}_{\text{absent};\,k}), a_{\text{best}}; \theta_{\text{final}}), \tag{46}$$

where θ_{final} is the parameterisation of the online network of the Nature DQN at the end of training, and where

$$a_{\text{best}} = \max_{a \in \mathcal{A}} Q\Big(\text{F}\big(\mathbf{s}_{\text{present};\,k}\big), a; \theta_{\text{final}}\Big). \tag{47}$$

Intuitively, $\Delta_k > 0$ suggests an action-value-heightening effect of k on the overall estimation of $\mathbf{s}_{\text{present};\,k}$'s expected return; vice-versa for $\Delta_k < 0$. Throughout the text, we have occasionally used the term 'Q-value delta' for the Δ_k.

If we use this interpretation to understand Figure 8's 'Pass' scene for the game of *Pong*, then the opponent's paddle (to the left) appears to provide a slight, positive action-value-heightening effect on the total action-value of the state derived from the scene.

In that same scene, we observe the ball having a considerably high Q-value delta. In particular, the color of the ball is a darker green than displayed on the color range. An inverse effect can be observed for some objects with significantly negative Q-value deltas; these have a dark-brown color instead of the color range's orange. These object instances illustrate our third and last point on interpreting our OSM plots: outlier objects. Object instances with a Q-value delta preceding or surpassing the minimum or maximum of the color range, respectively, are marked by placing a border around each pixel of the object instance's 'silhouette', producing these darker colors, implementing our second improvement proposal to Li et al. (2017)-style OSMS. In terms of interpretation, we may say that these object instances have a relatively large effect on the final state's Q-value, compared to non-outlier objects.

Experiment 2

Average undiscounted return curves. In terms of episode-averaged evaluation phase undiscounted return curves, experiment 2 diverges from experiment 1 in two regards. First, we use three repetitions to obtain our return curves. As a result we may compute the standard error around the curve per each iteration, which we show using a shaded region similar to the one used for the baseline. Second, we introduce the Rainbow agent, which causes our complete plot to be extended with a second row to accommodate for this addition. Of course, the most central difference to look out for in the plots is the exclusive use of object channels. Hence, a comparison with experiment 1's return curves may give insight in the effects of removing 'raw' greyscale screen input, and relying solely on what was previously only an

Table 3
An overview of the approximate positions of objects per game, per scene, in the OSMs (Figures 8, 10, and 13). Coordinates are given in the format (height, width)*, with both entries expressed in pixel indices. Entries with only a single coordinate pair imply a static object, invariant to three scenes; entries marked by a '–' communicate that that particular object cannot be found in that scene. Two causes are (I) the irregular intervals at which objects are present on screen, even when working with maximum-pooling, and (II) occlusion, preventing template matching from working correctly. Apart from this, note that we have omitted one object type—the regular Ms. Pac-Man pellets—due to their abundance. They can be identified as horizontally elongated rectangles in the Ms. Pac-Man OSMs.*
Note that one can verify these positions for themselves by comparing these coordinates and the OSMs with the scene source frames (Figures C1 and C2 in the Appendix). When doing so, be careful to take into account the resizing and max-pooling that took place to produce the OSMs.

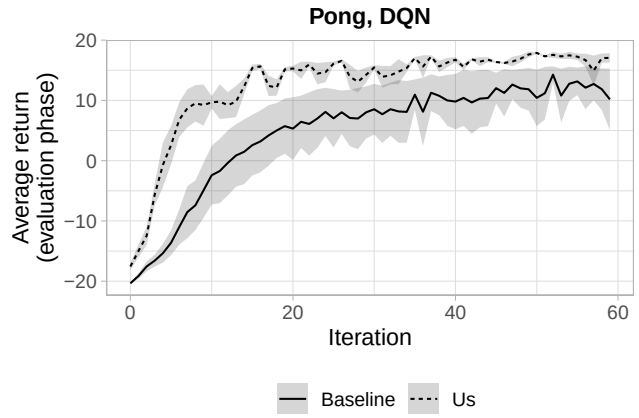
	Scene			
Pong	Lose	Pass	Win	Description
Our paddle	(62, 75)	(30, 75)	(30, 75)	The player's (rather, agent's) paddle.
Opponent's paddle	(30, 10)	(20, 10)	(25, 10)	The computer opponent's paddle.
Ball	(28, 80)	(17, 50)	(28, 10)	The ball.
Ms. Pac-Man	Pellets	Power pellet	Crossroad	Description
Ms. Pac-Man	(41, 38)	(74, 10)	(62, 22)	The player's (agent's) character.
Blinky	(62, 38)	(80, 22)	(70, 6)	The first ghost to enter the maze.
Clyde	(50, 46)	(68, 70)	(62, 62)	The second ghost to enter the maze.
Inky	(45, 46)	–	(53, 41)	The third ghost to enter the maze.
Pinky	–	(50, 38)	(48, 38)	The fourth ghost to enter the maze.
Power pellet 1		(80, 80)		The upper-right power pellet.
Power pellet 2		(22, 80)		The lower-right power pellet.
Life 1		(12, 8)		The leftmost (last-to-be-used) life of Ms. Pac-Man.
Life 2		(12, 16)		The rightmost (first-to-be-used) life of Ms. Pac-Man.

input-side augmentation. Figure 9, then, presents the average undiscounted return curves for experiment 2.

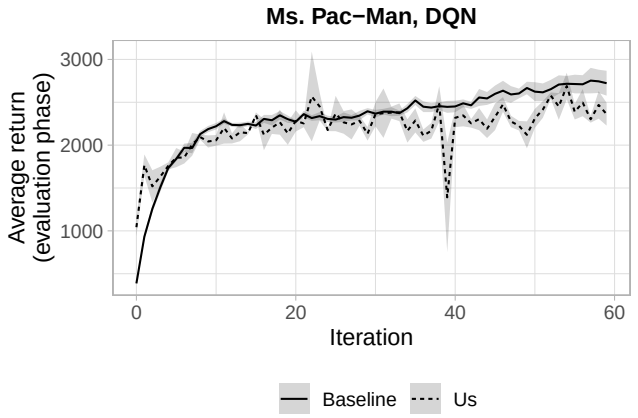
The left column displays the object channel-only versions of the Nature DQN and Rainbow applied to *Pong*; they are shown in Figures 9a and 9c, respectively. Both plots suggest that the object only-versions of the agents, again, reach the same level of average undiscounted return, albeit at a relatively 'early' iteration. If we use a 'convergence score' of 17 points, then the object channel-only Nature DQN seems to reach this average undiscounted return around 40 iterations, while the baseline—again referring to the extended baseline plot provided by (Castro et al., 2022)—requires 130 iterations. Similarly, let us choose a convergence score of 20 points for the Rainbow agents. Then the object channel-only agent appears to reach this target around the 15$^{\text{th}}$ iteration, whereas the baseline requires almost 100 iterations; Figure 9c even suggests that the baseline has already converged to a policy that produces average undiscounted returns that structurally score a point lower than the object channel-only counterpart.

Perhaps more interesting is the right column, which presents the same two agents' return curves, except on *Ms. Pac-Man* instead of *Pong*. Again, the upper Figure, Figure 9b, shows the object channel-only Nature DQN, while Figure 9d below it presents Rainbow. Recall experiment 1's return curves, and especially our comment on how it appeared that our object channel-augmented Nature DQN agent yielded worse average undiscounted returns in the last few iterations, compared to the baseline. Figure 9b seems to show a more extreme trend of this kind: in the last third of the iterations, the object channel-only Nature DQN produces lower average undiscounted returns than the baseline. Still, it does not seem as if the object channel-only Nature DQN stagnates, as in the last five iterations the curve increases lightly again. These observations are in stark contrast to what can be seen in Figure 9d, directly below it. There, after approximately iteration 4, the object channel-only Rainbow agent's average undiscounted return curve departs from the trend shown by the baseline Rainbow agent, with at its peak near iteration 25 a difference in score of around 1,000 points. Moreover, while the object channel-only Rainbow agent seems to reach its 'convergence score' of nearly 4,000 points around iteration 40, the baseline Rainbow agent requires almost 180 iterations (Castro et al., 2022).

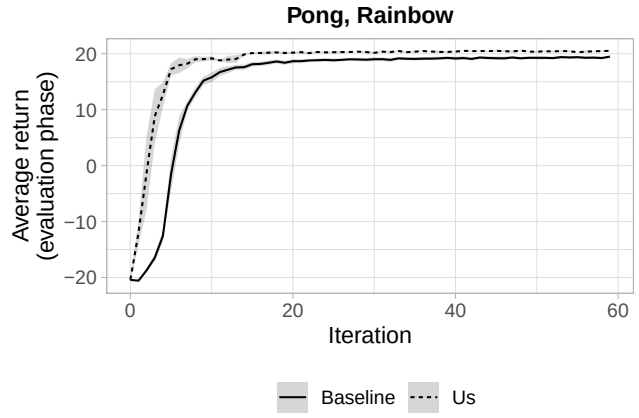
With the use of three repetitions, we may gain a better insight in the level of noise around our 'grand averages' of average undiscounted returns. For *Pong*, the object channel-only



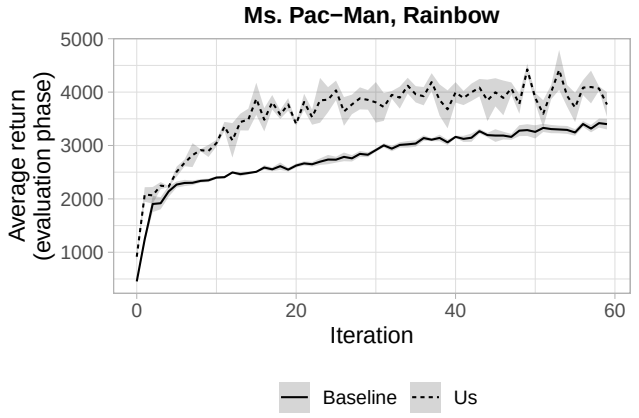
(a) Nature DQN–*Pong*.



(b) Nature DQN–*Ms. Pac-Man*.



(c) Rainbow–*Pong*.



(d) Rainbow–*Ms. Pac-Man*.

Figure 9. Episode-averaged undiscounted return curves for experiment 2, varying over the two games *Pong* and *Ms. Pac-Man*, and also considering both the Nature DQN and Rainbow as DRL models. Both the baselines and our results display 'grand averages' and standard errors of individual episode-averaged undiscounted return curves. The baselines used 5 repetitions, while we used 3.

agents seem to produce curves with similar levels of noise as the baselines—perhasp even less oscillatory in the specific case of the Nature DQNS. However, the object channel-only agents applied to *Ms. Pac-Man* do appear to yield more varying levels of average undiscounted return, whether we consider the Nature DQN or Rainbow agents.

Object saliency maps. In Figure 10, we present the object saliency maps for our second experiment. As is the case with the average undiscounted returns under experiment 2, we add extra rows to the Figure to accommodate for Rainbow's production of OSMS, besides the Nature DQN. Similarly, we average over each DRL model-video game's three OSMS to obtain aggregated OSMS, as laid out in the Methodology.

Due to the averaging operation, our interpretation of the 'OSMS'—in actuality aggregated OSMS—needs to subtly change. Specifically, while previously Equation 46 was appropriate for singular object instance Q-value deltas, we now need to adapt it to

$$\overline{\Delta}_k = \sum_{i=1}^{3} \Delta_{k;i}, \tag{48}$$

where the iteration, in the general case, of course need not be restricted to three repetitions. $\overline{\Delta}_k \in \mathbb{R}$ is the average Q-value delta over the three repetitions' individual Q-value deltas, $\Delta_{k;1}, \Delta_{k;2}, \Delta_{k;3} \in \mathbb{R}$. As a direct consequence, we should add the qualification 'on average' before interpreting singular object instance averaged Q-value deltas.

Experiment 3

Object segmentation masks. Because the unsupervised MOREL subnetwork is so central to the environment's observation pipeline in experiment 3, it may be informative to

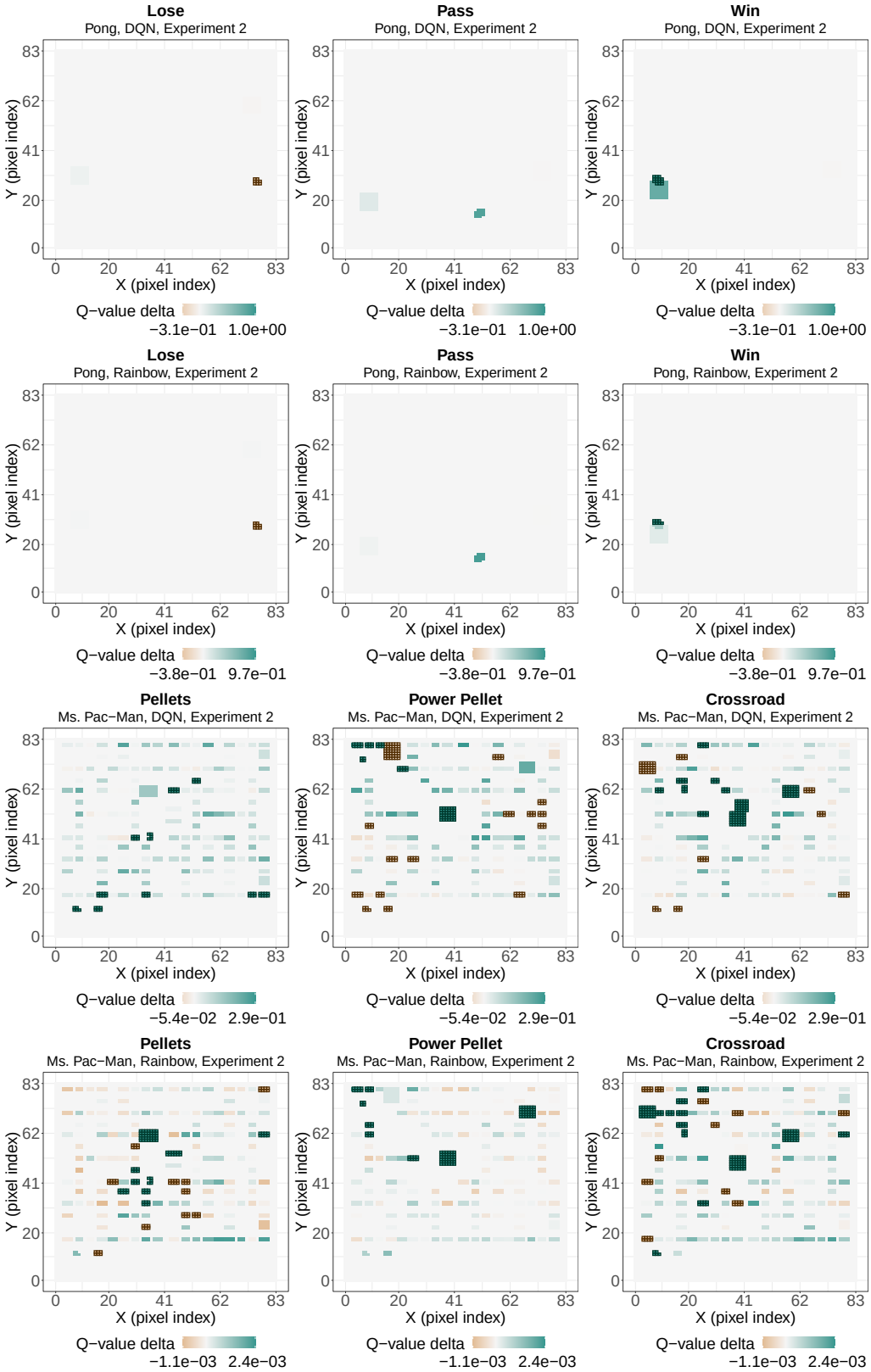


Figure 10. Object saliency maps for experiment 2.

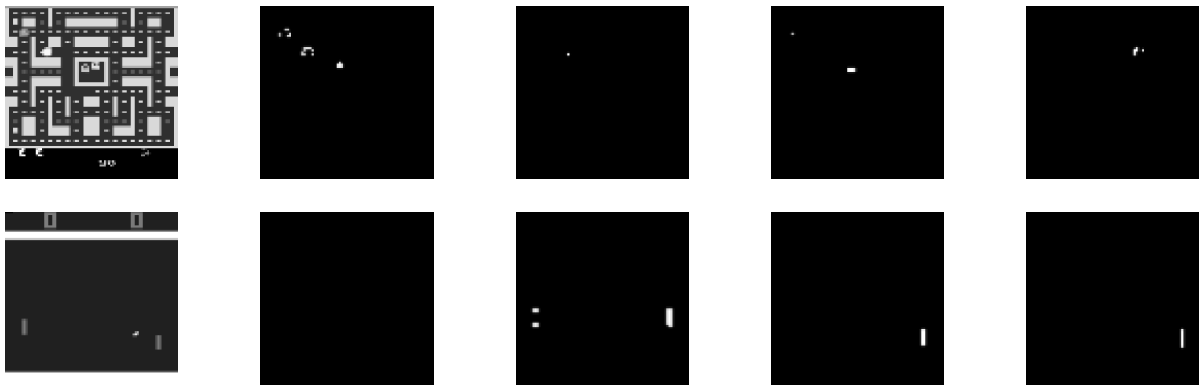


Figure 11. Two examples of object segmentation mask results. The first row shows a relatively successful example of object segmentation via motion. The second row demonstrates that, sometimes, not all moving objects in the scene are captured by MOREL's unsupervised subnetwork. Note that the displayed frames are processed. Hence, they are of 84×84 pixels, and can seem somewhat blurry.

consider the quality of output generated by this subnetwork before moving on to the return curve and OSM discussions.

In Figure 11 we show two examples of the $K = 4$ object segmentation masks produced by the subnetwork. The top row shows the network trained on *Ms. Pac-Man*, and applied to the 'Crossroad' scene, whereas the bottom row displays the network trained on *Pong* and confronted with the 'Pass' scene. We have chosen these two exemplary output masks, as they are representative of the qualitative behaviour of the network, also in different scenes.

Specifically, we observe that the MOREL subnetwork learns to partially segment the image's objects. In both rows of Figure 11, we can discern a strict subset of the objects from the Li et al. (2017)-style object channels being segmented: for *Ms. Pac-Man*, it appears that Blinky, Ms. Pac-Man, and a ghost in the maze's centre are captured throughout the four masks of the top row. Similarly, both the player's as well as the opponent's paddles seem to be encoded in the object segmentation masks of the network of the bottom row of the Figure.

Now, we emphasise the word 'partially' from before, because arguably relevant objects are still missing from the masks. Although one may state that *Ms. Pac-Man*'s object segmentation masks are relatively complete—perhaps the blinking power pellets could still be captured—the encoding in *Pong* is notably incomplete: it does not store a segmentation of the ball in any of its four channels, while such information is probably essential to successful gameplay. Despite the incompleteness of the resultant object segmentation masks, however, it is still interesting to consider to what degree the two DRL models considered in this thesis may leverage this imperfect source of additional information.

Besides this main observation, we report on successfully reproducing two qualitative phenomena mentioned by Goel et al. ((2018), pp. 6–7).

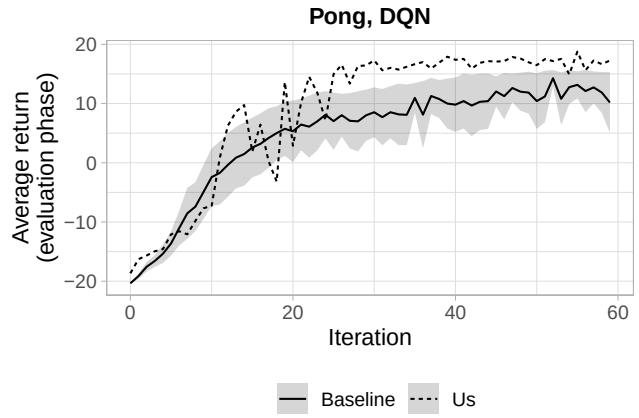
The first phenomenon is due to the 'low-texturedness' of Atari 2600 games. Particularly, if we translate a uniformly-colored block over the vertical or horizontal axis, and if the destination position overlaps at least partially with the starting position, then the MOREL subnetwork estimates for the overlapping pixels a movement of zero pixels, because technically these pixels do not change brightness. This can be seen in the lower row of Figure 11: the opponent's paddle consists of two small blocks, which is the 'difference' in the paddle over the considered skip cycle.

Second, in relatively 'object-intensive' Atari 2600 video games, multiple objects tend to move consistently in similar directions—or exactly the same direction. Goel et al. (2018) brought up the example of *Space Invaders*, in which the enemy ships move in a coordinated manner. In our Figure 11, we see in *Ms. Pac-Man*'s row that the first object channel—which should normally consist of a single, unique object—actually contains both Blinky, Ms. Pac-Man, as well as the maze-centre ghost. It seems that, in this situation, the network has learned to perform an 'aggregate motion prediction' on all three objects.

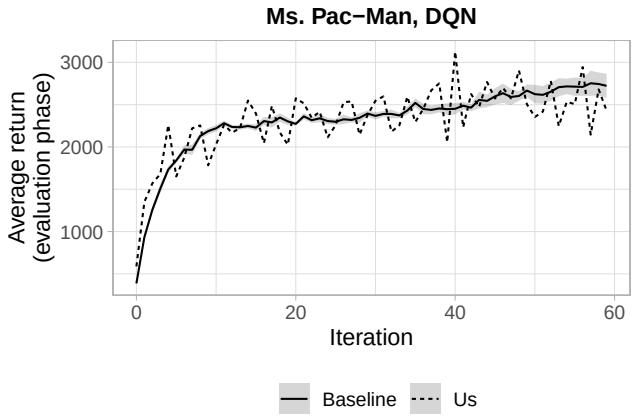
Having briefly inspected the MOREL subnetwork used to generate the motion-component of our environment's observations, we now move on to the average return curves.

Average undiscounted return curves. For experiment 3, we again show the performance in terms of episode-averaged evaluation phase undiscounted returns for the Nature DQN and Rainbow agents, applied on *Pong* and *Ms. Pac-Man*. Different from experiment 2 is that we revert back to one repetition due to computational limitations, as was already explained within the Methodology. Figure 12 presents the average undiscounted returns for experiment 3.

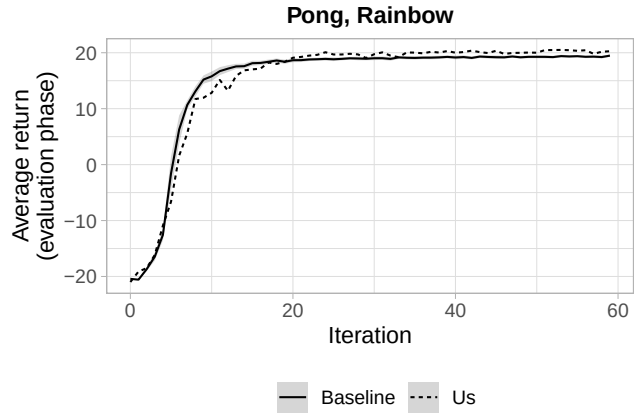
As we did before, we concentrate first on the left column,



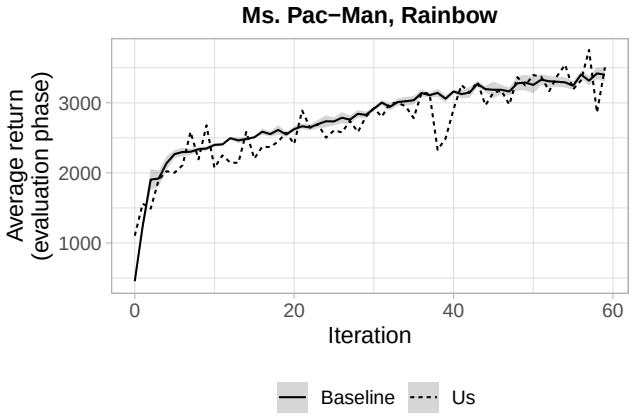
(a) Nature DQN–*Pong*.



(b) Nature DQN–*Ms. Pac-Man*.



(c) Rainbow–*Pong*.



(d) Rainbow–*Ms. Pac-Man*.

Figure 12. Episode-averaged undiscounted return curves for experiment 3, varying over the two games *Pong* and *Ms. Pac-Man*, and also considering both the Nature DQN and Rainbow as DRL models. The baselines' interpretation remains the same. Note that we revert back to one iteration in comparison to experiment 2, so we directly show an individual episode-averaged undiscounted return, and no standard error, as in experiment 1.

which summarises the performances of the greyscale-and-motion Nature DQN and Rainbow agents on the game of *Pong* (Figures 12a and 12c, respectively). Interestingly, we observe two apparent stages in the two agents' average undiscounted return curves. The fist stage occurs within iterations 0 up until approximately 20. There, the return curves of the greyscale-with-motion agents seem to fall below those of the baseline counterparts. Directly after the first stage, there appears to be a 'turning point' after which the greyscale-and-motion agents produce average undiscounted returns that seem to lie below the baseline return curves. At the final iteration we considered, we see, similar to experiment 2, that the baseline curves require an additional number of iterations to reach the same level of average undiscounted reward. Since the 'convergence scores' of both the greyscale-and-motion Nature DQN and Rainbow lie around the values they took on during experiment 2, we omit reporting on the baselines' number of iterations to reach those scores, as they would be identical to those reported in experiment 2.

To the right, in Figures 12b and 12d, we do not discern the two-stage pattern. Instead, for both the greyscale-and-motion Nature DQN and Rainbow agents, the average undiscounted return over iterations for the game *Ms. Pac-Man* seems to progress similarly over iterations as for the baseline agents, although our two agents again display more variation around the central average undiscounted return. As was the case with *Ms. Pac-Man* during experiment 1, it may be that in the last few iterations (from iterations 55–60) the greyscale-and-motion Nature DQN agent performs slightly worse than the baseline counterpart, but we would need to run for a prolonged number of iterations to become more confident in this observation.

Lastly, we notice that for *Pong* our greyscale-and-motion Nature DQN and Rainbow agent show similar levels of oscilla-

tion in average undiscounted return as the iterations progress; for *Ms. Pac-Man* this is not so, as we discussed in the preceding paragraph.

Object saliency maps. Figure 13 displays the object saliency maps obtained from experiment 3. Since we return to only using a single repetition, our interpretation from experiment 1 can be re-used. Unlike experiment 1 however, we derive additional OSMs using Rainbow agents as well.

Examination of object saliency maps

We end the Results section by taking a closer look at the three experiments' object saliency maps. Particularly, we point to commonalities and differences between the OSMs at the experiment-, video game-, and agent-level, instead of describing each OSM in words, as this would demand too much space and would arguably duplicate in writen form what is already displayed visually.

Commonalities. We begin with considering what is shared among the OSMs, across the three experiments, two DRL models, and two video games.

First, we concentrate on *Pong*. What can be noticed across all OSMs is that the ball consistently assumes the most extreme Q-value deltas and average Q-value deltas of all three objects portrayed in the maps: in the 'Lose' scene the ball is the most negative of all three, whereas in 'Pass' and 'Win' it is most positive, or at least on par with the opponent paddle. In direct contrast is our own paddle, which takes on near-zero Q-value deltas (or averages of those) across the experimental configurations. Indeed, in some OSMs the paddle is nearly invisible, as it takes on the neutral Q-value delta of 0 of the background. Between these two extremes lies the opponent's paddle, which takes on relatively pronounced or subdued Q-value deltas, depending on the scene. Generally, it appears that in the 'Pass' and 'Win' scenes, the paddle is associated with relatively large Q-value deltas—although not as large as the ball—while the opposite applies to the 'Lose' scene. An exception here is experiment 2's Rainbow agent applied to the 'Pass' scene, in which the opponent's paddle is nearly zero in terms of Q-value delta.

Moving on to *Ms. Pac-Man*, we notice five patterns.

First, the titular character takes on a positive Q-value delta (or average Q-value delta) in nearly all experimental configurations, large to such a degree that these are marked as outliers. The only deviation is the 'Crossroad' scene supplied to the Nature DQN agent in experiment 1, where Ms. Pac-Man instead has a light-negative Q-value delta.

Similarly, her leftmost, last life is also frequently assigned a positive Q-value delta or average of Q-value deltas. Two counterexamples exist in experiment 2, where the Nature DQN assigns it a negative value instead. Also, in experiment 3 with the Rainbow–'Crossroad' scene combination, the life is nearly neutral.

Ms. Pac-Man's rightmost, first life is instead consistently changing Q-value delta 'polarity' across experiments, DRL models, and scenes. Comparing experiment 2's Nature DQN–'Power pellet' value with experiment 3's Rainbow–'Pellets' value illustrates this point.

Fourth, we observe that in nearly every single OSM both positive and negative Q-value delta pellets can be found, although often pellets with a shared polarity are located directly adjacent to one another. Experiment 2's Nature DQN–'Pellets' configuration forms an exception, as there almost all pellets are positively-valued.

Fifth and last, the ghosts inhabiting the contraption at the maze's centre are given high Q-value deltas (or averages thereof), even to an extent that they are sometimes marked as positive outliers. This does not hold in the OSM produced by the Rainbow agent on the 'Power pellet' scene of experiment 3, though.

Differences. We continue to the differences among the various OSMs. Our coverage of this topic is split in three parts. We first consider broad differences observable between the three experiments. Then, video game-specific differences are considered. Finally, we close our examination of the OSMs by listing some deviations due to using a different DRL model.

At the level of experiments, observe how the *Pong* OSMs of experiments 2 and 3 are nearly identical in terms of (average) Q-value delta polarities and relative magnitudes. Instead, in experiment 1 we see a discrepancy: the two paddles take on negative (average) Q-value deltas in two of the three scenes. For *Ms. Pac-Man* it is more challenging to find structural differences between the three experiments. One pattern that seems to apply is that in experiment 2, none of the scenes across any of the two DRL models contains multiple dozens of outlier-level negative averaged Q-value deltas. We do observe this in experiments 1 and 3—consider the Nature DQN applied to 'Pellets' for the former, or Rainbow–'Power pellet' for the latter. Related to this is that it appears as if relatively large Q-value deltas, regardless of polarity, concentrate closely to *Ms. Pac-Man* in experiment 2, while such values are scattered more evenly over the maze in experiments 1 and 3.

Between games, we note that *Pong* has relatively large Q-value deltas, or averages thereof, in comparison with the deltas for *Ms. Pac-Man*. Specifically, when comparing any experiment–DRL model combination, the associated Q-value delta range is always shifted downwards when moving from *Pong* to *Ms. Pac-Man*; this is a relatively small difference of $1.2 \cdot 10^{-1}$ for experiment 1, but takes on a double-magnitude difference in experiments 2 and 3.

A similar downward shift can be observed when comparing DRL agents instead of games: the Rainbow agents' Q-value delta ranges (or ranges of averages of these deltas) are shifted downward from $3 \cdot 10^{-2}$ points up until a difference of, again, two magnitudes.

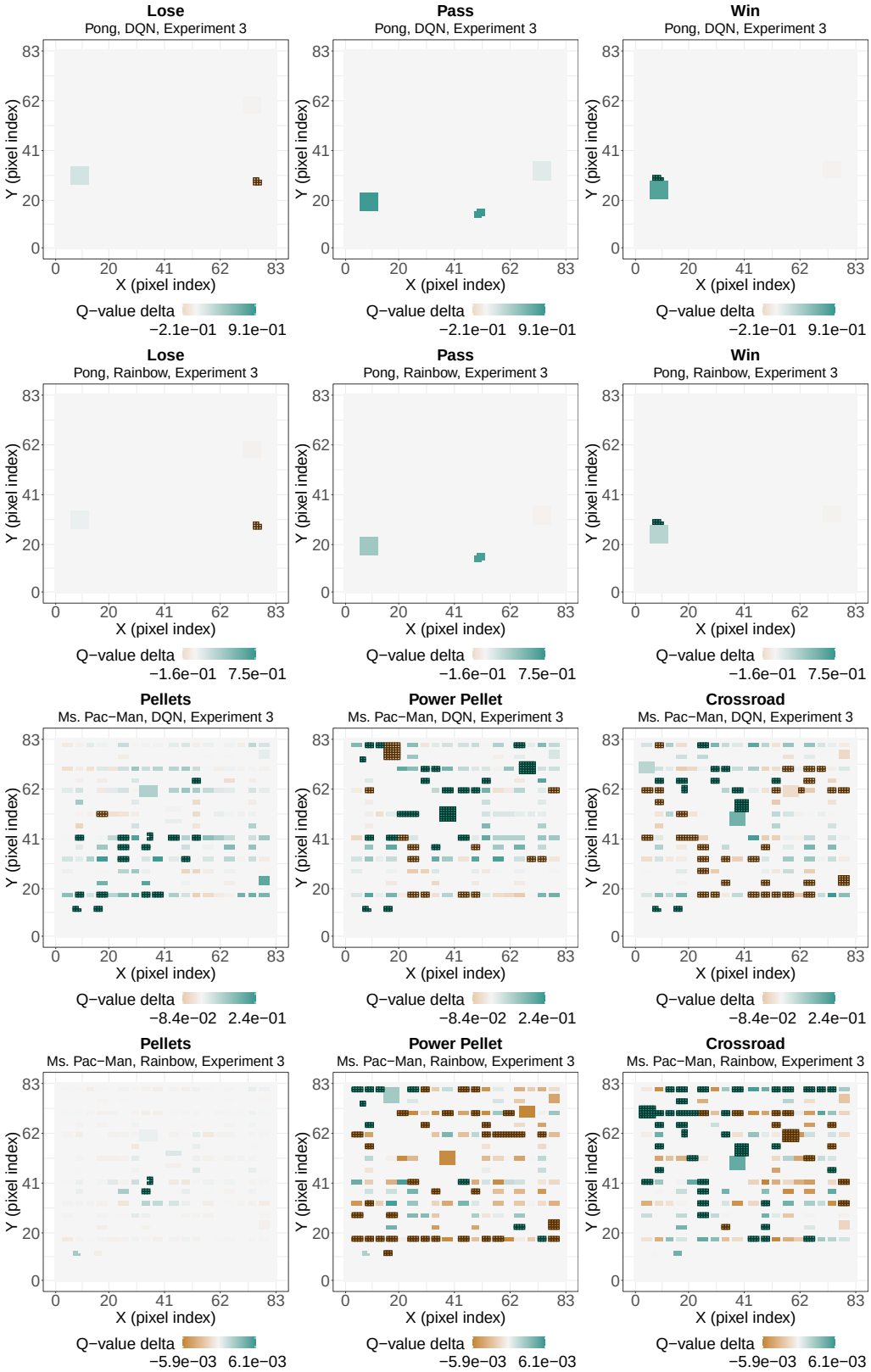


Figure 13. Object saliency maps for experiment 3.

Discussion

Recall from the Methodology that we have designed our methods in such a way so that we may answer the two research questions from the Introduction by considering the Results' outcomes across all three experiments. As such, we discuss the results by first considering the three experiments' average undiscounted return curves, and then the object saliency maps, instead of stepping through all three experiments separately. Of course, if any noticeable effects take place when shifting from one experiment to another, we remark on these.

Average undiscounted return curves

We start with the average undiscounted return curves. What can be noticed across all three experiments is that the majority of all object-augmented DRL models appear to reach certain levels of average undiscounted return faster than their baseline counterparts, in terms of number of iterations required. An important exception is the Nature DQN model applied to *Ms. Pac-Man*, which only succeeded in reaching average undiscounted returns faster during experiment 2. Furthermore, Rainbow only seems to have managed to match the average undiscounted return curve for the baseline when using inputs consisting of 'raw' greyscale screens and object segmentation masks (experiment 3).

An interesting follow-up question is to ask how much faster the object-augmented systems reach these average undiscounted return levels. Table 4 attempts to answer this question by aggregating from the Results, per each experiment, video game, and model, how long the object- and non-object versions of a model took to reach a set average undiscounted return: the 'target AURS' in the Table.

As can be derived from Table 4, if the object-augmented model indeed improves over the baseline, we see speed-ups of 70% up until 85%, which is considerable. Of course, there are two important qualifications that need to be made with this statement. First, the potential speed-ups that may be expected can vary per game and per DRL model type. This is illustrated by the difference in the Nature DQN versus Rainbow on *Ms. Pac-Man*: only the latter agent type appears to manage consistently to obtain a speed-up, or at least, no degradation in learning speed. Second, the selection of target AUR may impact the precise speed-up value. We have chosen the first point at which the average undiscounted return curve surpasses the targeted average undiscounted return. Instead, one might reasonably argue that the curve should have surpassed the target for some number of iterations, and stay repeatedly above it during that period.

As Goel et al. (2018) already suggest, the observed speed-ups may be attributed to an increased sampling efficiency, or stated differently, a reduction in sampling complexity. 'Sampling' refers here to drawing quadruples $(\text{F}(\mathbf{s}_{t'}), a_{t'}, r_{t+1}, \text{F}(\mathbf{s}_{t'+1}))$ from the experience replay buffer $\mathcal{B}$, while 'efficiency' and 'complexity' are opposing extremes that point to the same attribute of samples. Intuitively put, highly efficient (low-complexity) samples represent the same 'information' with relatively little 'noise' compared to inefficient (complex) samples. Consequently, DRL models learning from high-efficiency samples may concentrate more on improving their parameterisation to relate input features to output action-value estimates. Models learning from inefficient samples additionally need to to learn to ignore input noise while establishing the relation. This requires extra sample presentations without improving the mapping from states to action-value estimates. Hence, an increased sampling efficiency—or, equivalently, a reduced sampling complexity—may contribute to accelerated learning.

In the above, 'information' refers to aspects of the input that may be successfully used to improve the DRL model's mapping from inputs to action-value estimates, while 'noise' is any apparent pattern that cannot be structurally leveraged for said mapping's improvement. Thus, a sample's observations often contains both information and noise, regardless of whether said observations are made from greyscale screen inputs or object channels. The suggested possibility is that the information-to-noise ratio of observations using object channels may be higher than those using greyscale screen input, arguably because object channels present the observations in a highly abstracted and minimal form.

Another pattern found within the Results is that the Nature DQN agent consistently appears to fail to improve over the baseline in terms of learning speed, while this is not so for the Rainbow agent. For this phenomenon, we propose the following explanation. From the argument on sample efficiency above, it may be that the information-to-noise ratio of baseline inputs—only greyscale screens—is lower than that of object channel inputs; the former may contain relatively many elements that are irrelevant to estimating action-values correctly, such as, possibly, the color of a maze's background in *Ms. Pac-Man*. However, the more 'object-intensive' an observation becomes, the lower the benefit of using object channels over greyscale screens becomes, as the latter now has a higher information-to-noise ratio, while the former's information-to-noise ratio remains approximately the same: in the object channels, noise was already minimal. Now, depending on the performative capacity of a DRL model, this smaller benefit for more object-intensive games may or may not be leveraged, or worse: it may demand more time steps for the agent to learn the relation of states to action-value estimates, as there are more channels to learn from. It seems safe to say that *Ms. Pac-Man* is more object-intensive than *Pong*: the number of pellets in *Ms. Pac-Man* alone already surpass the number of total objects in *Pong*, Thus, perhaps in *Ms. Pac-Man*, only the relatively performant Rainbow agent may leverage the object channels' information effectively.

One last matter we discuss regarding the Results' average

Table 4
An overview of speed-ups—percent point reductions in number of iterations required to reach the targeted average undiscounted return (AUC)—when using object-augmented DRL models instead of baseline models provided by Castro et al. (2018), for each experiment, video game, and model. '–' indicates that an object-augmented model performs inferior to the baseline, and that as such there is no speed-up.

Experiment	Game	Model	Target AUR	With object channels	Without object channels	Speed-up (%)
1	*Pong*	Nature DQN	15	30	120	75.0
	Ms. Pac-Man	Nature DQN	–	–	–	–
2	*Pong*	Nature DQN	17	40	130	69.2
		Rainbow	20	15	100	85.0
	Ms. Pac-Man	Nature DQN	–	–	–	–
		Rainbow	4,000	40	180	77.8
3	*Pong*	Nature DQN	17	40	130	69.2
		Rainbow	20	20	100	80.0
	Ms. Pac-Man	Nature DQN	–	–	–	–
		Rainbow	3,400	60	60	0.0

undiscounted reward curves is that seemingly, using greyscale screens plus object segmentation masks, we obtain agents that learn less rapidly than the object-based agents relying on Li et al. (2017)-style templates.

We hypothesise that this gap may be caused by a difference in 'consistency' between template-based and object segmentation mask-based object channels. By this we mean that the two approaches differ in (I) how often objects are detected, and (II) whether they are presented in a similar form across frames. If the input is highly consistent in this sense, then a learning DRL model may learn the input-output mapping faster because the anomalous cases of absent or irregularly-detected objects need not be regularly accounted for; vice-versa for inconsistent object channels. Then, template-based object channels are possibly more consistent because (I) they detect objects regardless of presence or degree of motion, (II) they produce objects of a uniform, rectangular shape—object segmentation masks may deliver varying shapes of 'blobs', as can be seen in Figure 11, and (III) they are relatively robust to shifts in the input distribution, while object segmentation masks are, preferably, adapted continuously alongside the agent during training (Goel et al., 2018, p. 5,690).

Object saliency maps

As we explained in the Methodology, we are not in the position to address our second research question in full. Instead, we critically consider whether the OSM may serve as an object-based model explainability tool, and, based on this, provide only a tentative answer. At the end of this subsection, we additionally present a possible operationalisation for evaluating the OSM's merit as an explanatory tool.

Critical evaluation of OSMs. We present two arguments in favour of the potential use of OSMs as an object-based model explainability tool, and two arguments against.

The first affirmative argument is relatively straightforward, and appeals to how Q-value deltas may intuitively appeal to the human notions of 'good' and 'bad', as also already suggested in (Li et al., 2017)'s original work (p. 5). It may further be seen as an improvement over the pixel saliency map (Simonyan et al., 2013) as it presents 'good' or 'bad' influences in a model's determination of Q-values at the level of objects, which is arguably a more natural level of resolution for humans to reason at than single pixels.

The second argument for the use of OSMs as an explainability tool is based on the empirical data we have collected over the three experiments. Specifically, we see that multiple important objects in both *Pong* and *Ms. Pac-Man* have Q-value deltas that probably align with intuitions that expert and non-expert human users would have about those objects. For instance, the Q-value deltas that the ball in *Pong* takes on in the situations 'Lose', 'Pass', and 'Win' appear logical: the ball is negatively-valued when it is certain that it will cause a point for the opponent; it is generally positive when moving towards the opponent (i.e. there is no 'risk' of a loss), and it has an outlier-level positive Q-value delta when it is certain that the player will score a point. Similarly, we would arguably expect Ms. Pac-Man and the leftmost (last) life to be valued positively. Reasoning inductively from such empirical observations, we could say that the OSMs produce at least for a subset of the objects human-interpretable and -explainable results.

These empirical observations may simultaneously also be used to argue against the appropriateness of OSMs as an object-

based model explainability tool, as we may also find examples of objects that are valued contrary to intuitions that end-users (or even expert users) may have about the likely Q-value deltas of such objects. For instance, the rightmost (first) life in *Ms. Pac-Man* is only sometimes assigned a positive Q-value delta, while at other times it is negatively-valued. Similarly, regular pellets vary widely in their Q-value deltas—from highly positive to highly negative. Now, one can propose intricate arguments as to why these objects are valued the way they are. Absence of a pellet, for instance, may suggest that template matching at its location failed due to a ghost obscuring the pellet, thus justifying the 'unintuitive' Q-value delta. The problem with these hypotheses is that we must perform further testing to confirm or reject them, defeating in part the purpose of the tool. Worse, if no further investigation is performed, the resultant hypotheses stand as-is, without testing. This makes OSMs prone to 'informed guessing', while, preferably, explainability tools are unambiguous and clear in their interpretation.

There is one other potential weakness of OSMs that may be important. Notice that, at least in our implementation, OSMs are always built by using the Q-value maximising action a_{best} in the original state $\mathbf{s}_{\text{present}}$, and keeping this a_{best} constant when collecting Q-values for the various 'absent states', $\mathbf{s}_{\text{absent}}$. Now consider the following situation. We remove some object from the original state, producing a specific $\mathbf{s}'_{\text{absent}} \in \mathcal{S}$. It is not completely inconceivable that the Q-value for $\mathbf{s}'_{\text{absent}}$ is differing minimally from the Q-value of $\mathbf{s}_{\text{present}}$. In these cases, we would obtain a near-neutral Q-value delta. Now consider how the other actions' Q-values may change from the exclusion of the object under consideration. It may happen that another action's Q-value changes considerably positively—perhaps even becoming the action-value maximising action in $\mathbf{s}'_{\text{absent}}$. This is possible, because the action-values for a state are not 'mutually inhibitory', as, for example, the entries of a SoftMax layer (Appendix A) are. Surely, this change is important with respect to explaining the causes to a model's decision, but it is not captured in the OSM.

If we collectively consider the two arguments for and against the application of OSMs to improving model explainability, then the final conclusion remains undecided. OSMs may intuitively appeal to expert and end-users alike. Still, their empirical results in the three experiments gives mixed results depending on which objects we concentrate on specifically. We have also identified two obstacles that prevents OSMs from being interpreted 'naively', as the maps may deliver ambiguous results that require additional testing. This aligns with the discussion presented by Atrey, Clary, and Jensen (2020), which argues that saliency maps ought to be regarded as exploratory tools instead of explanatory ones.

Further research on OSMs. In order to determine whether object saliency maps may work as an explanatory tool for DRL models, we must first realise that important work has already been performed by Iyer et al. (2018) (pp. 147–149). Specifically, Iyer and colleagues tested for the game *Ms. Pac-Man* (I) whether OSMs can be matched with game scenes, similar to those we have provided in Appendix C, (II) whether participants could produce 'reasonable' explanations for the behaviour of Ms. Pac-Man in said scenes, and (III) whether the participants could predict how Ms. Pac-Man would behave after the one-to-last frame. They subsequently designed psychological tests around these three questions, and asked human participants to perform these tests. By expanding these designs to include the object channel-only and greyscale-and-motion agent types, and by conducting tests on multiple Atari 2600 video games and scenes, the second research question may be answered more rigorously. Of course, this is only one suggested approach.

Conclusion

In this thesis, we have attempted to answer two research questions, one regularly, the other only tentatively:

1. Does representing the state by its high-level objects accelerate learning in deep reinforcement learning methods, and

2. Can these high-level object representations make deep reinforcement learning methods more explainable?

Given the results and our discussion around them, it appears that the first question may be answered affirmatively, provided that the specific DRL method is sufficiently effective to leverage the additional information provided by the objects, and given that these object-level representations are presented consistently. Regarding potential gains in explainability provided by object-level representations, we have critically examined (Li et al., 2017)'s object saliency map applied to the three methods considered in this thesis. The conclusion of said examination was neither definitely affirmative nor negative—a further investigation into the procedure is required instead, for which we have suggested one possible approach.

We hope that this thesis has motivated the reader, along with others within the deep learning community, to (re-)consider the possible utility of high-level objects in DRL methods specifically and DL methods generally, be it in terms of accelerating learning, or potentially improving explainability of current methods—both arguably central topics at the current frontier of artificial intelligence.

Acknowledgements

Access to the Peregrine supercomputing cluster at the University of Groningen has been invaluable to obtain timely results, for which the author would like to thank its supporting staff.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... Google Brain (2016). TensorFlow: A System for Large-Scale Machine Learning. In K. Keeton & T. Roscoe (Eds.), *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation* (Vol. 12, pp. 265–283). Retrieved from https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Adadi, A. (2021). A Survey on Data-Efficient Algorithms in Big Data Era. *Journal of Big Data*, *8*(24). doi: 10.1186/s40537-021-00419-9

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., & Kim, B. (2018). Sanity Checks for Saliency Maps. In *Advances in Neural Information Processing Systems* (Vol. 31, pp. 9,505–9,515). Retrieved from https://proceedings.neurips.cc/paper/2018/hash/294a8ed24b1ad22ec2e7efea049b8737-Abstract.html

Atrey, A., Clary, K., & Jensen, D. (2020, December 9). *Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning*. arXiv pre-print. Retrieved from https://arxiv.org/abs/1912.05743

Baird, L. C. (1993, November 4). *Advantage Updating* (Technical Report). Dayton, OH, United States: Wright-Patterson Air Force Base. (The internal technical report identifier is WL-TR-93-1146.)

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th International Conference on Machine Learning* (Vol. 70, pp. 449–458). Retrieved from https://proceedings.mlr.press/v70/bellemare17a.html (The volume and URL pertain to the submission of this article in the Proceedings of Machine Learning Research.)

Bellemare, M. G., Naddaf, Y., & Bowling, J. V. M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, *47*, 253–279. doi: 10.1613/jair.3912

Bellman, R. (2010). *Dynamic Programming* (1st ed.). Princeton University Press.

Bradski, G. (2000, November). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, *25*(11), 122–125. Retrieved 5 June 2022, from https://www.drdobbs.com/open-source/the-opencv-library/184404319

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016, June 5). *OpenAI Gym*. arXiv pre-print. Retrieved from https://arxiv.org/abs/1606.01540

Buşoniu, L., Babuška, R., de Schutter, B., & Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators* (1st ed.). CRC Press.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., & Bellemare, M. G. (2018, December 14). *Dopamine: A Research Framework for Deep Reinforcement Learning*. arXiv pre-print. Retrieved from https://arxiv.org/abs/1812.06110

Castro, P. S., Vassalotti, A., Greaves, J., Agarwal, R., Evci, U., Moitra, S., ... Teboul, O. (2022, May 20). *dopamine*. GitHub. Retrieved June 7, 2022, from https://github.com/google/dopamine (The latest commit at our time of reference was a34615cb611c7c23b63ca7e6fee45331e6e1f912. Further, the baselines we refer to throughout the main text can be viewed at https://google.github.io/dopamine/baselines/atari/plots.html.)

Chong, E., Han, C., & Park, F. C. (2017). Deep Learning Networks for Stock Market Analysis and Prediction: Methodology, Data Representations, and Case Studies . *Expert Systems with Applications*, *83*, 187–205. doi: 10.1016/j.eswa.2017.04.030

Clark, A., & Contributors. (2010, July 31). *Pillow*. Image processing library. Retrieved 6 June 2022, from https://python-pillow.org/

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., ... Legg, S. (2018). Noisy Networks for Exploration. In Y. Bengio, Y. LeCun, T. Sainath, I. Murray, M. Ranzato, & O. Vinyals (Eds.), *Proceedings of the International Conference on Learning Representations* (Vol. 6). Retrieved from https://openreview.net/forum?id=rywHCPkAW

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., & Kagal, L. (2018). Explaining Explanations: An Overview of Interpretability of Machine Learning . In F. Bonchi & F. Provost (Eds.), *International Conference on Data Science and Advanced Analytics* (Vol. 5, pp. 80–89). Turin, Italy: IEEE.

Goel, V., Weng, J., & Poupart, P. (2018). Unsupervised Video Object Segmentation for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems* (Vol. 31, pp. 5,683–5,694). Retrieved from https://proceedings.neurips.cc/paper/2018/hash/96f2b50b5d3613adf9c27049b2a888c7-Abstract.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* (1st ed.). MIT Press.

Goodfellow, I., Shlens, J., & Szegedy, C. (2014, December 20). *Explaining and Harnessing Adversarial Examples* . arXiv pre-print. Retrieved from https://

arxiv.org/abs/1412.6572

Guo, W., Dong, G., Chen, C., & Li, M. (2019). Learning Pushing Skills Using Object Detection and Deep Reinforcement Learning. In *IEEE International Conference on Mechatronics and Automation* (Vol. 16, pp. 469–474). doi: 10.1109/ICMA.2019.8816481

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020). Array Programming with NumPy. *Nature*, *585*, 357–362. doi: 10.1038/s41586-020-2649-2

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., ... Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In S. A. McIlraith & K. Q. Weinberger (Eds.), *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, pp. 3,215–3,222). Palo Alto, CA, United States: AAAI Press.

Hinton, G., Srivastava, N., & Swersky, K. (2012). *Neural Networks for Machine Learning: Lecture 6e: RMSProp: Divide the Gradient by A Running Average of its Recent Magnitude.* Presentation Slides. Retrieved June 1, 2022, from https://www.cs.toronto.edu/%7Etijmen/csc321/slides/lecture_slides_lec6.pdf

Horn, B. K. P., & Schunk, B. G. (1981). Determining Optical Flow. *Artificial Intelligence*, *17*(1–3), 185–203. doi: 10.1016/0004-3702(81)90024-2

Huber, P. J. (1964). Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, *35*(1), 73–101. doi: 10.1214/aoms/1177703732

Iyer, R., Li, Y., Li, H., Lewis, M., Sundar, R., & Sycara, K. (2018). Transparency and Explanation in Deep Reinforcement Learning Neural Networks. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (Vol. 1, pp. 144–150). doi: 10.1145/3278721.3278776

Jain, P., Hosch, W. L., & other Encyclopedia Britannica contributors. (2011, August 11). Atari Console [Encyclopedia article]. In *Encyclopedia Britannica.* Retrieved May 25, 2022, from https://www.britannica.com/technology/Atari-console

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the Best Multi-Stage Architecture for Object Recognition? In T. Matsuyama, R. Cipolla, M. Hebert, X. Tang, & N. Yokoya (Eds.), *IEEE 12 International Conference on Computer Vision* (pp. 2,146–2,153). doi: 10.1109/ICCV.2009.5459469

Johnson, M., Hawkins, P., Vanderplas, J., Necula, G., Frostig, R., Wanerman-Milne, S., ... Klitgaard, J. (2022, July 7). *jax*. GitHub. Retrieved July 7, 2022, from https://github.com/google/jax (The latest commit at our time of reference was 2b8fbe9fe4753dce17d4093b5e1a70db9feb164a.)

Kahneman, D. (2011). *Thinking, Fast and Slow*. Macmillan.

Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization. In Y. Bengio, Y. LeCun, B. Kingsbury, S. Bengio, N. de Freitas, & H. Larochelle (Eds.), *Proceedings of the International Conference on Learning Representations* (Vol. 3). Retrieved from https://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* (Vol. 25, pp. 1,097–1,105). Retrieved from https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Lample, G., & Chaplot, D. S. (2017). Playing FPS Games with Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence* (Vol. 31, pp. 2140–2146). Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14456/14385

Li, Y., Sycara, K., & Iyer, R. (2017). Object-sensitive Deep Reinforcement Learning. In *Global Conference on Artificial Intelligence* (Vol. 3, pp. 20–35). doi: 10.29007/xtgm

Lin, L.-J. (1992). Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, *8*, 293–321. doi: 10.1007/BF00992699

Lundh, F. (1995). *Python Imaging Library.* Image processing library. Retrieved from http://www.pythonware.com/products/pil/ (Accessed on 6 June 2022 using the Wayback Machine (https://web.archive.org/), by selecting the date 21 November 2020.)

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2018). Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, *61*, 523–562. doi: 10.1613/jair.5699

Marcus, G. (2018, January 2). *Deep Learning: A Critical Appraisal.* arXiv pre-print. Retrieved from https://arxiv.org/abs/1801.00631

Mira, J. M. (2008). Symbols Versus Connections: 50 Years of Artificial Intelligence. *Neurocomputing*, *71*, 671–680. doi: 10.1016/j.neucom.2007.06.009

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd International Conference on Machine Learning* (Vol. 48, pp. 1,928–1,937). Re-

trieved from https://proceedings.mlr.press/v48/mniha16.html (The volume and URL pertain to the submission of this article in the Proceedings of Machine Learning Research.)

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013, December 19). *Playing Atari with Deep Reinforcement Learning*. arXiv pre-print. Retrieved from https://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-Level Control Through Deep Reinforcement Learning. *Nature*, *518*, 529–533. doi: 10.1038/nature14236

Mohamed, A.-R., Dahl, G. E., & Hinton, G. (2012). Acoustic Modeling Using Deep Belief Networks. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 14–22. doi: 10.1109/TASL.2011.2109382

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st ed.). John Wiley & Sons.

Riedmiller, M. (2005). Neural Fitted *Q* Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method . In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, & L. Torgo (Eds.), *Machine Learning: European Conference on Machine Learning* (Vol. 3720, pp. 317–328). Springer. doi: 10.1007/11564096_32

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay. In H. Larochelle, S. Bengio, B. Kingsbury, Y. Bengio, & Y. LeCun (Eds.), *Proceedings of the International Conference on Learning Representations* (Vol. 4). Retrieved from http://arxiv.org/abs/1511.05952

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013, December 20). *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps* . arXiv pre-print. Retrieved from https://arxiv.org/abs/1312.6034

Smith, C. S. (2020, February 12). *The Future of ML: Unsupervised Learning, Reinforcement Learning, or Something Else?* Web article. Retrieved 12 June 2022, from https://blog.paperspace.com/the-future-of-ml/

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An Introduction* (2nd ed.). MIT press.

van Hasselt, H. (2010). Double *Q*-Learning. In *Advances in Neural Information Processing Systems* (Vol. 23, pp. 2,613–2,621). Retrieved from https://proceedings.neurips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html

van Rossum, G. (1995, April 10). *Python Tutorial* (Technical Report). Amsterdam, the Netherlands: Centrum voor Wiskunde en Informatica. (The internal technical report identifier is CS-R9526.)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 5,998–6,008). Retrieved from https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Vijayanarasimhan, S., Ricco, S., Schmid, C., Sukthankar, R., & Fragkiadaki, K. (2017, April 25). *SfM-Net: Learning of Structure and Motion from Video.* arXiv pre-print. Retrieved from https://arxiv.org/abs/1704.07804

Wang, H., Lei, Z., Zhang, X., Zhou, B., & Peng, J. (2019). A Review of Deep Learning for Renewable Energy Forecasting. *Energy Conversion and Management*, *198*, 111,799. doi: 10.1016/j.enconman.2019.111799

Wang, Z., Bovik, A. C., Sheikh, H., & Simoncelli, E. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. doi: 10.1109/TIP.2003.819861

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd International Conference on Machine Learning* (Vol. 48, pp. 1,995–2,003). Retrieved from https://proceedings.mlr.press/v48/wangf16.html (The volume and URL pertain to the submission of this article in the Proceedings of Machine Learning Research.)

Watkins, C. (1989). *Learning from Delayed Rewards* (PhD Thesis). King's College, Cambridge, Cambridge, United Kingdom.

Appendix A
Activation Functions

Below, we briefly explain what the activation functions used within the main text refer to. Before doing so, we must establish what we mean by the term. At least within this text, by an activation function, we mean a function mapping tensors to equally-shaped tensors

$$\mathrm{a} : \mathbb{R}^{L_1, \ldots, L_\mathrm{L}} \to \mathbb{R}^{L_1, \ldots, L_\mathrm{L}},$$

where $\mathrm{L} \geq 1$ is the number of dimensions of the input and output tensors, and where $L_1, \ldots, L_\mathrm{L} \geq 1$ are the number of entries contained within each of the dimensions of the tensor. For instance, for a 3×2 matrix, the general form of the activation function would be

$$\mathrm{a}_{3\times 2\,\mathrm{matrix}} : \mathbb{R}^{3\times 2} \to \mathbb{R}^{3\times 2}.$$

It must be noted that the output dimension $\mathbb{R}^{L_1, \ldots, L_\mathrm{L}}$ may, for certain activation functions, actually be more constrained than one real-valued output per input entry; we simply define it as such to cover all cases generally. One example of an activation function with a constrained output is the logistic sigmoid, to be discussed below.

Let us now discuss the activation functions used in this text.

Linear. The linear activation function simply returns the input tensor as-is, without manipulating said tensor's entries in any way. If $\mathbf{x} \in \mathbb{R}^{L_1, \ldots, L_\mathrm{L}}$ is our input tensor, then the linear activation function is thus defined as

$$\mathrm{a}_{\mathrm{linear}}(\mathbf{x}) \stackrel{\mathrm{def}}{=} \mathbf{x}.$$

Logistic sigmoid. The logistic sigmoid activation function, sometimes referred to as the 'sigmoid activation', is defined as

$$\mathrm{a}_{\mathrm{logistic\ sigmoid}}(\mathbf{x}) \stackrel{\mathrm{def}}{=} \frac{1}{1 + e^{-\mathbf{x}}}.$$

The logistic sigmoid is characterised by its sending of input entries to the range $[0, 1]$, for each input tensor entry. Deeply negative input values are close to 0; deeply positive input values lie close to 1. Logistic sigmoid activations may be used to create neural network layers that confer, per each tensor entry, a 'degree of applicability', with values closer to 1 indicating greater application. This is, however, only one use case of logistic sigmoids.

Rectified Linear Unit. The rectified linear unit (ReLU; Jarrett, Kavukcuoglu, Ranzato, & LeCun, 2009) is a piecewise-linear function that clamps all strictly negative entries within the input tensor to zero; all non-negative values are left as-is:

$$\mathrm{a}_{\mathrm{ReLU}}(\mathbf{x}) \stackrel{\mathrm{def}}{=} \max(\mathbf{0}, \mathbf{x}),$$

where the maximisation happens entry-wise, and $\mathbf{0}$ shares the size of $\mathbf{x}$. According to Goodfellow et al. (2016), it is a good default for neural network layers.

SoftMax. The SoftMax activation function takes—as the name implies—an input tensor and performs an operation that is similar in spirit to regular maximisation. Specifically, it distributes a single unit of activation across all entries of the tensor, assigning relatively large portions of the unit to relatively large entries in the tensor. As would be expected, the maximal entry (or entries) is (are) assigned the largest portion(s). Mathematically, the SoftMax is defined as

$$\mathrm{a}_{\mathrm{SoftMax};\,\mathrm{i}}(\mathbf{x}) = \frac{e^{\mathbf{x}_i}}{\sum_{i=0}^{L_1 \times \ldots, L_\mathrm{L}-1} \mathbf{x}_i},$$

where $i \in [0,\, L_1 \times \ldots,\, L_\mathrm{L} - 1]$ accesses a single entry of the input tensor $\mathbf{x}$ using zero-based indexing. It must be noted that the SoftMax can also be applied across subsets of the axes. In that case it is guaranteed that all entries across the axes subset sum to one in activation.

Appendix B
Timeout Proof

We need to prove that for any episodic environment, if (I) the implication described in Equation 45 holds for the environment, and (II) if $\mathrm{dur}(n) = T_{\mathrm{timeout}} + c'$ with $n \in \mathbb{N}$ and $c' > 0$ (such that $\mathrm{dur}(n) > T_{\mathrm{timeout}}$), then we can always pick

$$\begin{aligned} t &= T_{n-1} + T_{\mathrm{timeout}} - 1 \\ &\overset{(\mathrm{II})}{<} T_{n-1} + \mathrm{dur}(n) - 1, \end{aligned}$$

with $t \in \mathcal{T}$ which, at $t + 1$, already leads to the timeout state $\mathbf{s}_{\mathrm{timeout}}$. Then we can obtain $\mathrm{dur}(n) \not> T_{\mathrm{timeout}}$.

The proof breaks down into five steps, which go as follows:

Step 1. Begin by considering any $n \in \mathbb{N}$. Then let us pick $t = T_{n-1} + T_{\mathrm{timeout}} - 1$.

Step 2. Utilise the definition given in Equation 44 to obtain the following equality:

$$\begin{aligned} \mathrm{dur}(n) &\overset{(\mathrm{II})}{=} T_{\mathrm{timeout}} + c' \\ T_n - T_{n-1} &\overset{(\mathrm{Eq.\,44})}{=} T_{\mathrm{timeout}} + c' \\ T_n &= T_{\mathrm{timeout}} + c' + T_{n-1} \\ T_n &= (T_{n-1} + T_{\mathrm{timeout}} - 1) + 1 + c' \\ T_n &\overset{(\mathrm{Step\ 1})}{=} t + 1 + c'. \end{aligned}$$

Step 3. Realise that from the result of step 2 it directly follows that $t < T_n$, as $t < t + 1 + c'$.

Step 4. The two premises of Equation 45 now hold, and as such we may obtain its consequent. Particularly, we obtain $p(\mathbf{s}_{timeout}, 0 \,|\, \mathbf{s}_t, a) = 1$, for all $\mathbf{s}_t \in \mathcal{S}$ and all $a \in \mathcal{A}$. We conclude that at $t + 1$ we already enter terminal state $\mathbf{s}_{\mathrm{timeout}}$ with probability 1.

Step 5. From step 4 we know that $t+1$ is the 'true' final step, call it T_n^*. Then, any further time steps $t+2, \ldots, t+1+c'$ are 'excessive', in the sense that trajectory $n+1$ already is supposed to start at $T_{(n+1)-1}+1$ (see the Theory, paragraph on episodes), which should be $T_n^*+1 = (t+1)+1 = t+2$. Formulated differently,

$$\begin{aligned} T_n^* &= t+1 \\ &\overset{\text{(Step 1)}}{=} (T_{n-1} + T_{\text{timeout}} - 1) + 1 \\ &= T_{n-1} + T_{\text{timeout}}, \end{aligned}$$

plugging this into the RHS of dur (Equation 44),

$$\text{dur}^*(n) = T_n^* - T_{n-1} \quad = (T_{n-1} + T_{\text{timeout}}) - T_{n-1} = T_{\text{timeout}}, \tag{49}$$

and so $\text{dur}(n) \not> T_{\text{timeout}}$, as we instead derive $\text{dur}^*(n)$, evaluating to T_{timeout}.

Appendix C
Object Saliency Map Scenes

Here, we display the scenes used to compute object saliency maps (OSMs) for the games *Pong* and *Ms. Pac-Man*, as shown in the main text. For each figure, recall from the Methodology that a scene—at least in our implementation—consists of 6×2 frames: six skip cycles, of which we take the last two frames to maximum-pool over. We use six instead of four skip cycles due to an implementation detail; although not strictly necessary, we include them here for reproducibility purposes. Further, it may be helpful to view these images using a PDF viewer, in which one can zoom in.

With this addressed, the scenes for *Pong* and *Ms. Pac-Man* are shown in Figures C1 and C2, respectively.

Appendix D
Object Saliency Map Q-Value Delta Ranges

In this appendix subsection, we present per experiment–DRL model–video game combination a boxplot of all Q-value deltas observable across the scenes and repetitions. These figures motivated our choice for the minimum and maximum Q-value delta, per each experiment–DRL model–video game group of OSM plots. The boxplots can be reviewed in Figure D1.

We limit the boxplots to the 5$^{\text{th}}$ and 95$^{\text{th}}$ percentiles instead of plotting all data. This is done because, for some of the combinations, certain outliers are so pronounced that the boxplot gets 'compressed', preventing us from clearly seeing the first, second, and third quartiles clearly. Also, vertical striped bars indicate the 10$^{\text{th}}$ and 90$^{\text{th}}$ percentiles of the data; these were used as the minimum and maximum of the OSMs of the collection under consideration, respectively.

Appendix E
Additional Experimentation

Here, we show additional experiments that we have performed for the games of *Fishing Derby* and *Freeway*. For these two extra games, we have followed nearly the exact same workflow as described in the Methdology, except that (I) for all three experiments, we conduct only single runs—even for experiment 2, and (II) we only collect episode-averaged undiscounted return curves.

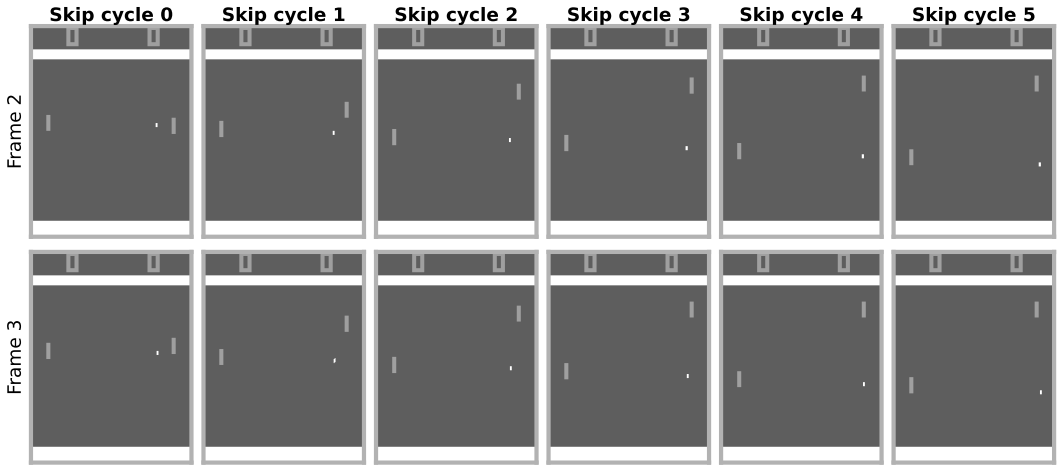
We first present the return curves, after which we briefly discuss how these results relate to what we found in the main results.

Results. Figures E1, E2, and E3 display the episode-averaged undiscounted return curves for experiments 1, 2, and 3, now applied to the games *Fishing Derby* and *Freeway*.

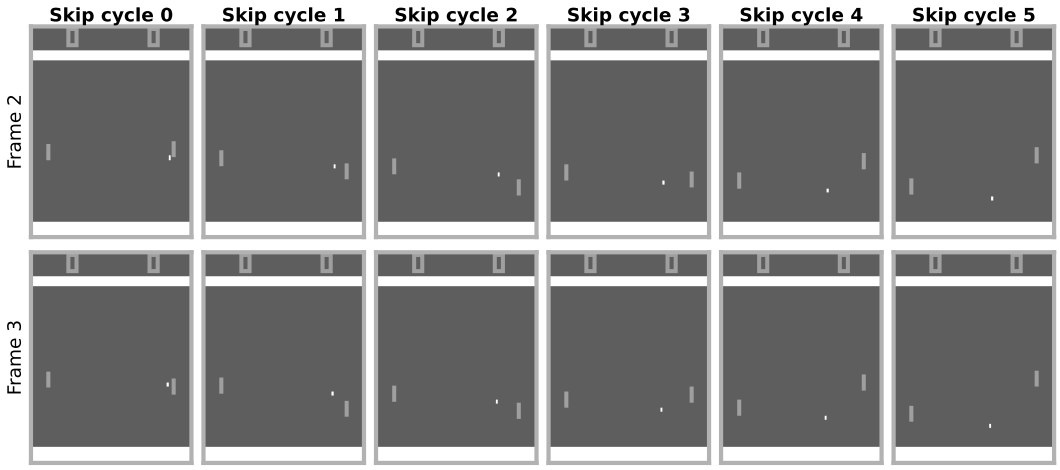
Let us begin by considering experiment 1 (Figure E1). In the left column, displaying results for *Fishing Derby*, the agent using both greyscale frames plus Li et al. (2017)-style object channels reaches a 'final' level of performance—a score of around 15 points—at approximately iteration 8 while the baseline attains a similar value nearly three iterations later. Furthermore, as the iterations progress, the object-augmented agent's average undiscounted returns seem to come closer and closer to the baselines. In the same Figure's right column, we see an almost opposite effect for the Nature DQN agent applied to *Freeway*. Particularly, the object-augmented agent appears to obtain average undiscounted returns (far) below the baseline for the major part of the return curve. Only from iteration 44 the situation changes: after its completion, the object-augmented agent's returns rapidly rise, even to a score seemingly above the baseline's maximal standard error value at the very last iteration. Important to notice here is that the object-augmented agent's score is not mostly below the baseline—it is perpetually zero.

Next, we look at experiment 2 (Figure E2), starting with the left column. The Nature DQN agent applied to *Fishing Derby* (Figure E2a) seems to closely mimic the return curve of Figure E1a: the 'final' average undiscounted return is attained comparatively early, after which the score remains stable, albeit with considerable noise over the iterations. The curve seems to, again, reach a score around 15 points, and this score is obtained before the tenth iteration. For the Rainbow counterpart in the plot below (Figure E2c), we again see that the object-augmented agent attains a score at an earlier iteration than the baseline—this time a score of approximately 7 points near iteration 5 instead of the baseline's iteration 9. Different from the Nature DQN agent, however, is that both the object-augmented and baseline return curves continue to improve after this value is reached: see iterations 10 and onward. Within this latter section of the curve, the object-augmented agent's return curve seems to match that of the baseline.

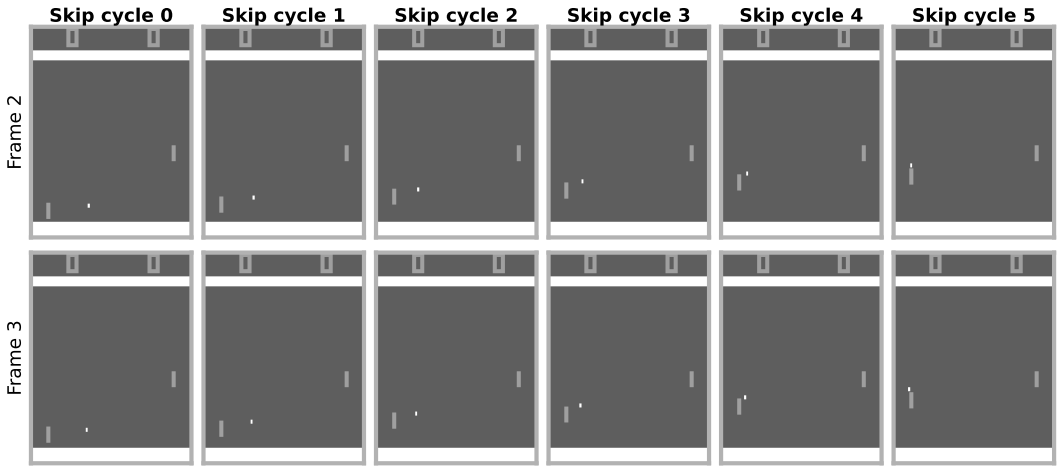
Continuing to the right column of Figure E2, which displays performances on *Freeway*, attention is arguably im-



(a) The ‘Lose’ scene: the ball gets behind our paddle (the right one).

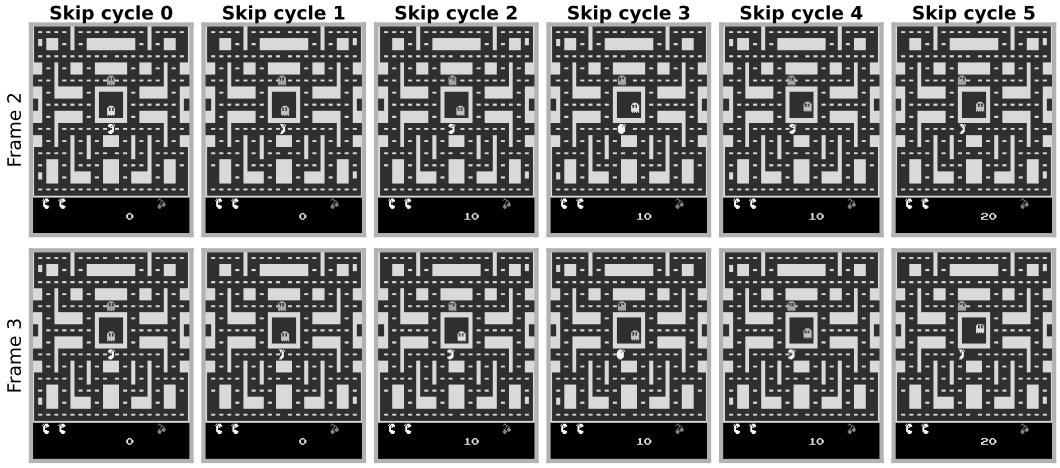


(b) The ‘Pass’ scene: the ball bounces from our side to that of the opponent.

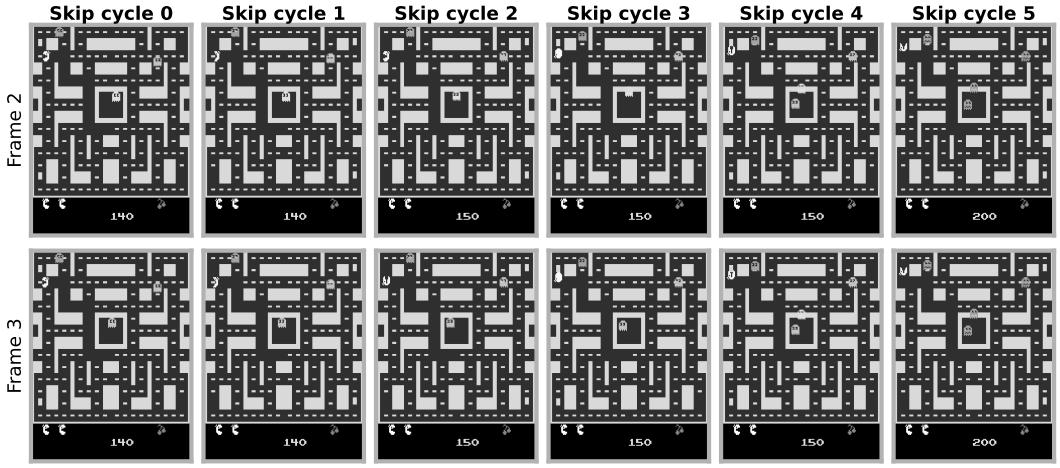


(c) The ‘Win’ scene: we score a point by getting the ball behind the opponent’s paddle.

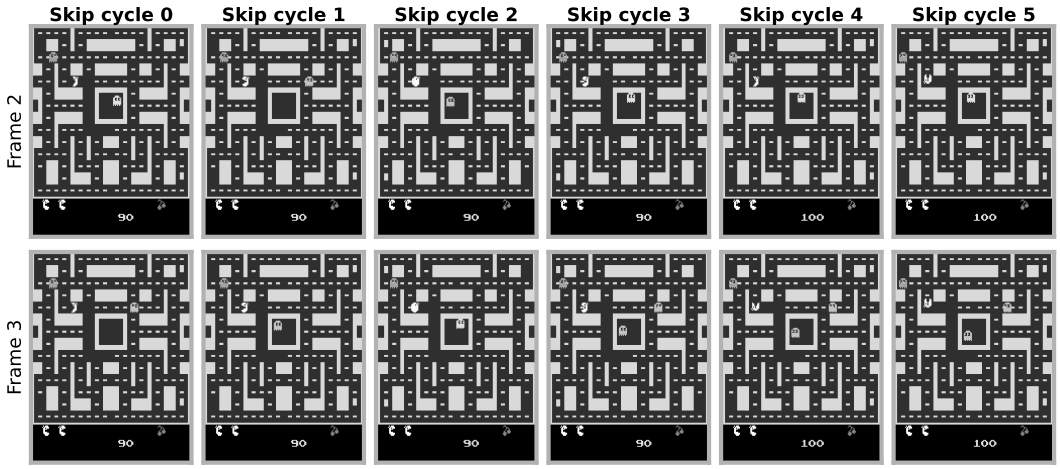
Figure C1. Scenes for *Pong*.



(a) The 'Pellets' scene: Ms. Pac-Man begins a new episode and eats the first few pellets.

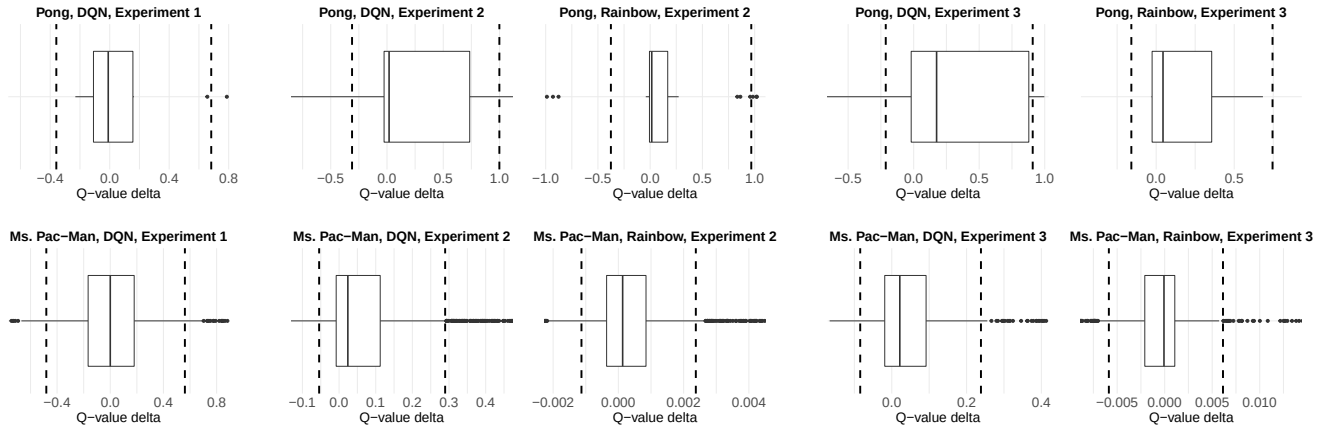


(b) The 'Power pellet' scene: Ms. Pac-Man eats a power pellet, making all normally-invincible ghosts vulnerable (by making them edible).



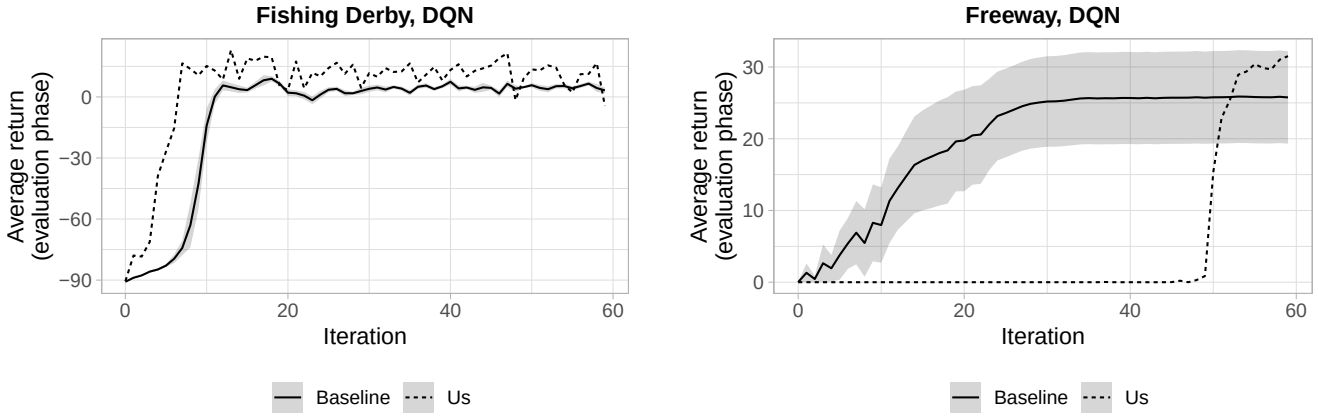
(c) The 'Crossroad' scene: Ms. Pac-Man needs to make a decision: at the upcoming juncture, should she turn left or right? On the left is a power pellet as well as Blinky (one of the ghosts) while the right side contains Clyde (another ghost) and various regular pellets.

Figure C2. Scenes for *Ms. Pac-Man*.



(a) Experiment 1. (b) Experiment 2. (c) Experiment 3.

Figure D1. *Q*-value delta ranges for the three experiments of the main text. In each figure, we show the 5$^{\text{th}}$ up until the 95$^{\text{th}}$ percentile of all *Q*-value deltas observed while creating the OSM for the experimental configuration under consideration. The striped vertical lines denote the 10$^{\text{th}}$ and 90$^{\text{th}}$ percentiles—the *Q*-value deltas at which we cut off the color bar range in the main text's OSM plots. Observe the values outside the 10–90$^{\text{th}}$ percentiles, but after and before the 5$^{\text{th}}$ and 95$^{\text{th}}$ percentiles, respectively: these are values marked as outliers in the OSMS, using per-pixel rectangles.

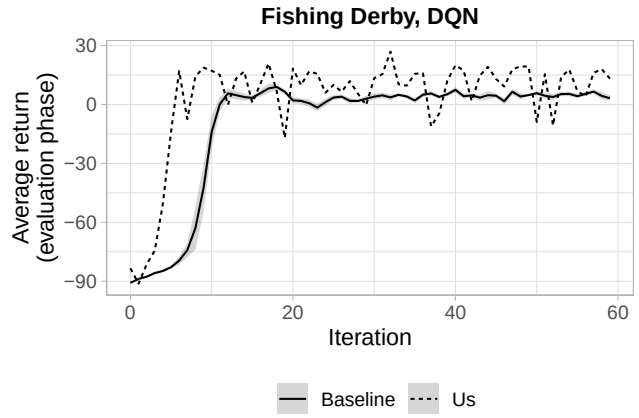


(a) Nature DQN–*Fishing Derby*. (b) Nature DQN–*Freeway*.

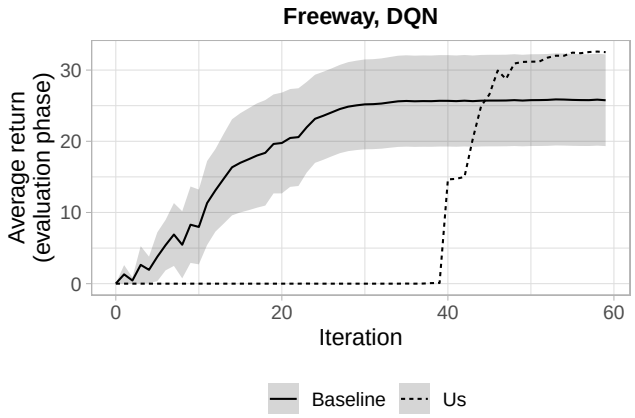
Figure E1. The average undiscounted return curves for experiment 1 on the games *Fishing Derby* and *Freeway*.

mediately drawn to the Nature DQN plot (Figure E2b). Similar to the *Freeway* graph from experiment 1, we see that the object-augmented agent's return curve lies considerably below the baseline for the major part of the curve. Only around iteration 39 the situation changes, similar to what we saw in Figure E1b. Different from experiment 1 is that the 'sudden switch' occurs earlier. As a consequence, we see a larger part of the return curve after the switch has completed—see iterations 55 and onward. From that point and further, it appears that the object-augmented agent stably remains at or just above the standard error of the baseline's return curve. The same effect cannot be observed in the Rainbow agent's return curve (Figure E2d); there, the baseline and the object-augmented agent seem to produce nearly indistinguishable average undiscounted returns over the iterations.

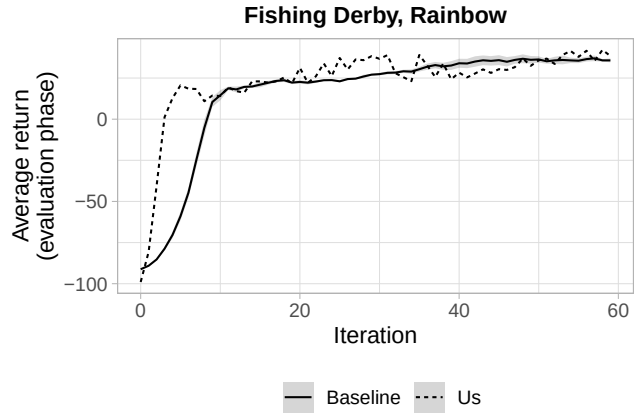
Last but not least are the return curves for the third experiment, shown in Figure E3). Again, we begin with the left column's upper-left graph, displaying the Nature DQN's average undiscounted returns on *Fishing Derby*. Different from the observations we made for the same configuration during experiments 1 and 2, here it seems that the object-augmented and baseline agents obtain similar average undiscounted returns across all iterations; notably, the object-augmented agent does not appear to reach the 'final' average undiscounted return at an earlier iteration than the baseline. Furthermore, the object-augmented agent obtains



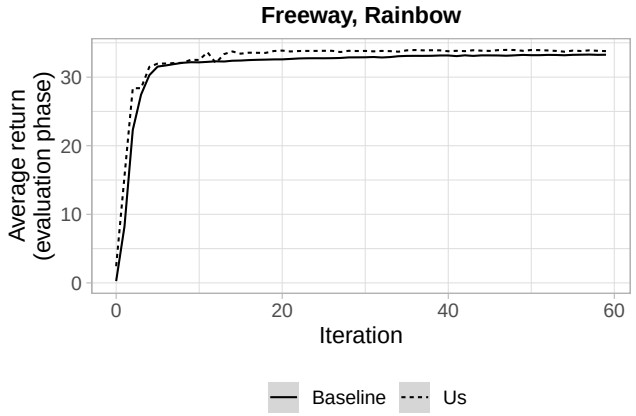
(a) Nature DQN–*Fishing Derby*.



(b) Nature DQN–*Freeway*.



(c) Rainbow–*Fishing Derby*.



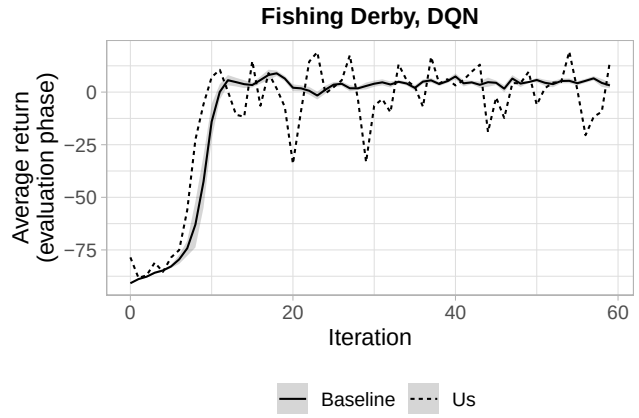
(d) Rainbow–*Freeway*.

Figure E2. The average undiscounted return curves for experiment 2 on the games *Fishing Derby* and *Freeway*. Note that, unlike the main results' experiment 2 return curves (displayed in Figure 9), we only use one repetition, not three.

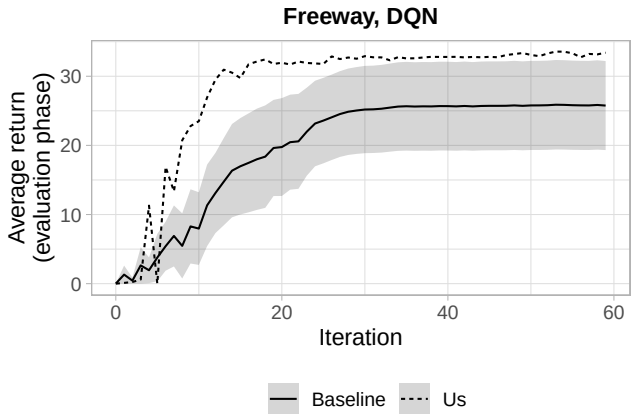
highly varying returns once this final score is reached, in contrast to the baseline. Below this plot we see the average undiscounted returns attained by the Rainbow agent. Qualitatively, the curve seems very similar to what we saw in experiment 2 (Figure E2c): the object-augmented agent's curve seems to reach a certain score at one or more iterations before the baseline reaches it, after which both curves follow a nearly identical trend. The difference with experiment 2 is that the 'headstart' of the object-augmented agent seems to be diminished: the difference is perhaps one or maximally two iterations whereas in experiment 2 this offset was closer to six or seven iterations.

On the right-hand side of the Figure, we notice that the Nature DQN agent deployed on *Freeway* now does seem to succeed in attaining a final average undiscounted return at an earlier iteration than the baseline; the 'convergence iterations' for the object-augmented and baseline agents seem to respectively be 15 and 25. Furthermore, the object-augmented agent reaches a final score that, as was the case in experiments 1 and 2, lies at or just above the baseline's upper standard error region. Moving to the Rainbow agent's curve below it (Figure E3d), we see an almost identical return curve to what could be seen in experiment 2's Rainbow–*Freeway* plot. This return curve nearly completely matches the baseline, with no clear offset being discernable.

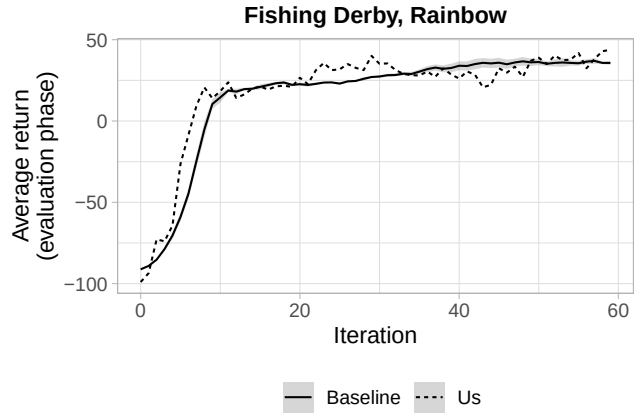
Discussion. Recall from the Results section of the thesis that, in a majority of the experimental configurations, the object-augmented agents appear to reach pre-set 'final' levels of average undiscounted return at earlier iterations than do the baseline counterparts. This conclusion also applies to the results we have collected here, although the 'majority' only consists of 6 out of 10 experimental configurations. Specifically, we argue that in all *Fishing Derby* experiments the object-augmented agents have at least a one-to-two-iteration lead on the baseline in terms of reaching near-final average undiscounted return; for *Freeway*, only the Nature DQN agent applied during experiment 3 seems to improve over the baseline in terms of accelerated learning. (For our definition of



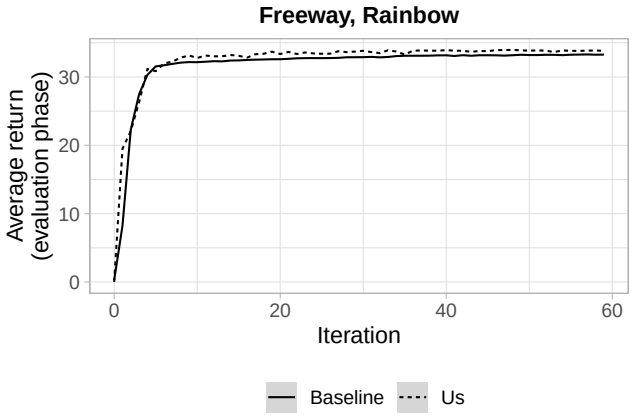
(a) Nature DQN–*Fishing Derby*.



(b) Nature DQN–*Freeway*.



(c) Rainbow–*Fishing Derby*.



(d) Rainbow–*Freeway*.

Figure E3. The average undiscounted return curves for experiment 3 on the games *Fishing Derby* and *Freeway*.

accelerated learning, review the Methodology from the main text.) All remaining experimental configurations seem to produce either (I) neither an advantage nor a disadvantage—these are all Rainbow agent plots, so this class consists of two configurations, or (II) they produce a clear disadvantage—exemplified in the Nature DQN plots for experiments 1 and 2, and so this class also consists of two configurations. All in all, the experimental outcomes from this additional set of experiments still supports the claim made in the thesis, although its degree of support appears weak.

The main text's reasoning behind the phenomenon of accelerated learning can be re-applied to the return curves we have obtained here. Similarly, we can put forth again the 'inconsistency' explanation when considering the relatively inhibited (but not non-existent) accelerated learning seen in the third experiment. Furthermore, considering that in *Fishing Derby* maximally eight objects are on-screen whereas in *Freeway* we may have twelve or even more, the argument of object complexity leading to the requirement of relatively capable networks—that is, Rainbow over the Nature DQN—may be plausible within our extended set of experiments as well.

What remains is one anomaly that was not observed in the main text's Results section: the phenomenon that the object-augmented agent's return curve lies considerably below the baseline until around the last third of the experiment—observable in the Nature DQN–*Freeway* scenarios for experiments 1 and 2.

In both Nature DQN–*Freeway* plots, we can see that once the return curves do attain a non-zero average undiscounted return at some iteration, the agent swiftly learns a policy similar to what was seemingly maximally attainable under the baseline. In experiment 1, this happens at iteration 44, while it occurs at iteration 39 in experiment 2. This suggests that the agent requires multiple tens of iterations before it has found a network parameterisation that produces Q-value estimates from which a policy can be derived that actually derives any points at all.

A possible explanation for this behaviour requires some understanding of *Freeway*. Put briefly, you, playing

as a chicken, need to cross the proverbial road as many times as you can while avoiding bypassing traffic. Points are only scored if the chicken reaches the end of the road, and you are moved back to the pavement at the bottom of the screen once you get hit by a car. There are ten lanes, with, on every lane, vehicles of varying widths moving by quickly. Now consider a randomly-initialised Nature DQN for this game. With arbitrary Q-value estimates for state-action pairs, the chance is low that the agent-controlled chicken successfully crosses even a single lane, let alone ten in succession. Problematic, then, is that zero reward is dispensed everywhere except when this low-probability event of a perfect road-crossing occurs. As a result, the parameterisation of the Nature DQN remains stuck on what is technically known as a (zero average undiscounted reward) 'plateau' in nonlinear optimisation. Only in the unlikely case that the agent, with the help of exploration, reaches the end in a single run, can it start to properly improve the network, escaping the plateau. This is what we see in the aforementioned iterations 44 and 39.

Still, this proposed explanation does not seem to apply exclusively to the object-augmented agent—why, then, do we not see a similar effect for the baseline? Actually, such an effect *can* be seen when considering the return curves provided by the Dopamine baselines (Castro et al., 2022) for related agent. Particularly, if we consider their Nature DQN agent, now, however, (I) trained using the Adam optimiser (Kingma & Ba, 2015), (II) corrected using the Mean Squared Error (MSE) instead of the Huber loss, and (III) implemented in JAX (Johnson et al., 2022), we see that similar 'late starts' happen there as well—the standard errors even suggest that some return curves may start later than iteration 45. All in all, it may perhaps be concluded that *Freeway* is generally a difficult game to learn for DRL agents, simply due to its design, and that probability plays a large role in determining when the agent starts learning properly. This, then, may interfere with our claims of accelerated learning.