Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Exploration-Exploitation Trade-off in Deep Reinforcement Learning

Bachelor Thesis


Lukas Rusch

ruschl@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich


**Supervisors:**
Oliver Richter
Prof. Dr. Roger Wattenhofer

August 31, 2018

# Acknowledgements

I would like to thank my supervisor Oliver Richter for his support throughout this thesis. His advice during our weekly meetings was always greatly appreciated.

# Abstract

A fundamental dilemma in reinforcement learning is the exploration-exploitation trade-off. Deep reinforcement learning enables agents to act and learn in complex environments, but also introduces new challenges to both exploration and exploitation. Concepts like intrinsic motivation, hierarchical learning or curriculum learning all inspire different methods for exploration, while other agents profit from better methods to exploit current knowledge. In this work a survey of a variety of different approaches to exploration and exploitation in deep reinforcement learning is presented.

# Contents

Chapter 1

# Introduction

Reinforcement learning agents learn autonomously by interacting with an environment. Deep reinforcement learning (DRL) agents utilize deep neural networks to approximate rich functions and learn useful representations from an agent's sensory inputs. Deep $Q$-learning [1] is famously able to learn to play many Atari 2600 games and achieve super-human performance on some, But deep reinforcement learning is still limited. In games where rewards are only sparsely awarded, an agent may fail to find any connection between its actions and the score it is trying to improve. In robotics, tasks are easiest specified by a single reward for successful completion, for example moving an object to a target location. DRL agents may also require millions of interactions with their environment, making them unsuitable for real world application where interactions can't be simulated.

Therefore, it is critical that agents can learn good features of the input space and interesting behaviors in the absence of rewards. Several approaches exist in the literature aiming to improve exploration by guiding them towards interesting regions in state space or to automatically learn to interact with the environment in new ways. Intrinsic motivation methods give an agent a human drive to explore and learn more about the environment, even if it does not directly lead to completion of a goal. Other exploration methods keep track of the uncertainty in the agents learning progress. Given DRL agents aim to directly learn and end-to-end policy from visual inputs, it has also proven beneficial to develop auxiliary techniques to efficiently find representations for the high dimensional inputs. Conversely to improved exploration techniques, a focus on exploiting the current policy has led to DRL algorithms with enhanced sample efficiency. A diverse set of methods addresses the problem of exploration and exploitation in DRL. This thesis presents an overview of the different approaches, as well as highlighting some specific approaches in more detail.

Chapter 2

# Background

## 2.1 Reinforcement Learning

The setup in reinforcement learning consists of an agent interacting with an environment in discrete time steps. At each time step $t$ the agent observes a state $s_t$, then the agent decides on an action $a_t$ according to his policy $\pi$. It reaches a new state $s_{t+1}$ and receives a reward $r_t$. This interaction loop can be seen in Figure 2.1. An agent's goal is to learn a policy which maximizes the cumulative rewards.

Formally the dynamics in an environment can be modeled as a Markov Decision Process. It consists of:

- $\mathcal{S}$ , the set of states.
- $\mathcal{A}$ , the set of actions.
- $P_0 : \mathcal{S} \to [0, 1] : P_0(s)$, the initial state distribution
- $P : (\mathcal{S} \times \mathcal{A} \times \mathcal{S}) \to [0, 1] : P(s' \mid s, a)$ , transition probabilities describing the dynamics between states.
- $r : (\mathcal{S} \times \mathcal{A}) \to \mathbb{R} : r(s, a)$ , the reward function.
- $\gamma \in [0, 1]$, a discount factor that trades of short term rewards for low $\gamma$ against long term rewards.

With these definitions a policy is a function $\pi : (\mathcal{S} \times \mathcal{A}) \to [0, 1] : \pi(a|s)$. It determines the probability of taking an action $a$ given some state $s$. Importantly the action only depends on the current state and not its previous state history. This does not hinder the agent's ability, since the transition from one state to the next doesn't depend on the previous history either.

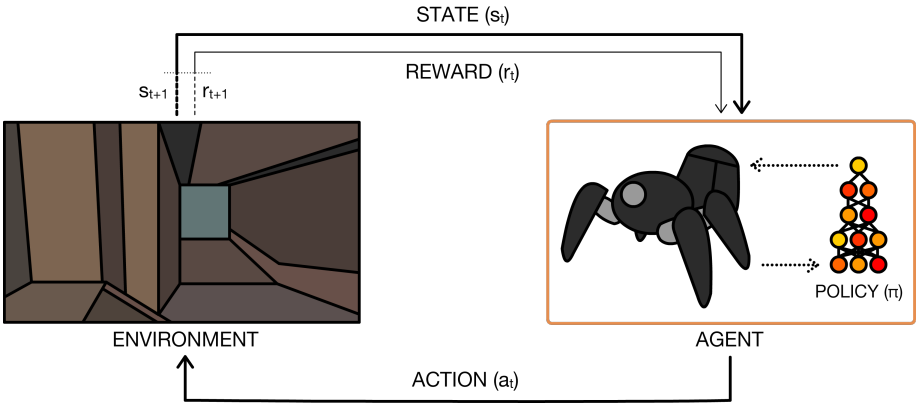


Figure 2.1: The perception-action-learning loop [2]0. At each time $t$, the agent observes a state $s_t$ from the environment. Then chooses action $a_t$ according to its policy $\pi$ and receives a reward $r_t$.

Consider a trajectory of observed states, actions and rewards $(s_0, a_0, r_0, s_1, ...)$. The return $R_t$ at state time $t$ is defined as $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ and the n-step return is $R_{t:t+n} = \sum_{i=0}^{n-1} \gamma^i r_{t+i}$. In environments such as video games the trajectory may not be infinite but end in a terminal state when the agent dies. In this case a trajectory from starting state to terminal state is called an episode. The goal of an agent is to find a policy which maximizes the expected return at each state $s_t$. The $Q$-value is the expected return of choosing action $a$ at state $s$ over the trajectories produced by some policy $\pi$.

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{(a_t, r_t, s_{t+1}, ..) \sim \pi} [R_t \mid s_t = s, a_t = a]$$

The value induced by policy $\pi$ is:

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [Q^{\pi}(s, a)]$$

For given $Q$-values the best policy is the greedy policy:

$$\pi(s) = \arg\max_{a} Q(a, s)$$

The Bellman equation can be used to calculate $Q$-values via bootstrapping for a given policy $\pi$:

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{s'} \left[ r(s, a) + \gamma Q^{\pi}(s', \pi(s')) \right]$$

The optimal $Q$-value $Q^*(s, a)$ is the maximum expected return given starting state $s$ and action $a$ and it satisfies the Bellman optimality equation:

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s'} [r(s, a) + \gamma \max_{a'} Q^*(s', a')]$$

Then the optimal policy is given by the greedy policy induced by $Q^*$.

## 2.2 Algorithms for Deep Reinforcement Learning

Deep reinforcement learning algorithms are employed in settings with complex environments. For example, learning video games from screen images or robotic control tasks. In these tasks agents must find efficient representations for the high dimensional inputs and be able to generalize between similar experiences.

### 2.2.1 Value-based algorithms

The goal of reinforcement learning algorithms is to find a policy which maximizes the expected return. In value-based algorithms this is done by learning the optimal values $Q^*(s, a)$ for each state $s$ and action $a$. This however is intractable for a large or continuous state and/or action space.

The deep $Q$-Learning algorithm uses a neural network to approximate a $Q$-value function. Consider some transition $(s, a, r, s')$ and a neural net with parameters $\theta$. The target is derived from the Bellman optimality equation:

$$y^{DQN} = r + \gamma \; max_{a'} Q(s', a'; \theta^-)$$

where $\theta^-$ are the parameters of a target network used for stability reasons.

Then the $Q$-value loss is given by

$$L_Q(\theta) = \mathop{\mathbb{E}}_{s'} (y^{DQN} - Q(s, a; \theta))^2$$

This loss is minimized with sampled transitions using stochastic gradient in the deep $Q$-learning algorithm 1. Here the difference between target and bootstrap estimate $y^{DQN} - Q(s, a; \theta)$ is called the temporal difference (TD) error.

With convolutional neural networks it is possible to train agents in complex visual domains without any handcrafted features [1]. But using a nonlinear function approximator like a neural network is unstable due to correlation between the samples, since they are drawn in sequence. This problem is addressed with a replay buffer that stores recent transitions. These transitions are then uniformly sampled to reduce correlations introduced by the otherwise sequential samples. To further improve learning stability a target network stores old parameters and is changed only periodically. The standard deep $Q$-learning algorithm employs a stochastic exploration strategy called $\epsilon$-greedy: The policy followed is the greedy policy according to the current $Q$-value estimate, but with probability $\epsilon$ a random action is chosen. This algorithm is model-free as neither the reward function $r$ or the transition probability is modeled. It is also off-policy as transition samples can be used, that are not generated with the current policy.

**Init:** $Q$-value network $Q$ with random weights $\theta$
**Init:** target $Q$-value weights $\theta^- = \theta$
**Init:** replay memory $D$, number of training episodes $M$, batch size $N_b$
**for** *episode* $e \in \{1, .., M\}$ **:**
  $s_0 \sim P_0(\cdot)$
  **for** $t \in \{1, ..\}$ **:**
    `/* Policy Rollout */`
    **if** *Ber(*$\epsilon$*)* **then** `/* random action with probability` $\epsilon$ `*/`
      $a_t \sim \text{Unif}(A)$
    **else**
      $a_t = \arg\max_a Q(s_t, a; \theta)$
    `/* Get next state and reward from environment */`
    $s_{t+1} \sim \text{P}(\cdot \mid s_t, a_t)$
    $r_t = r(s_t, a_t)$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
    `/* Training */`
    Sample a minibatch of $N_b$ transitions $(s, a, r, s') \sim \text{Unif}(D)$
    **if** $s'$ *is terminal* **then**
      $y = r$
    **else**
      $y = r + \gamma \; max_{a'} Q(s', a'; \theta^-)$
    Do gradient step wrt. to $\theta$ to minimize: $(y - Q(s, a; \theta))^2$
    Periodically update target network: $\theta^- = \theta$

**Algorithm 1:** Deep $Q$-Learning (DQN) Algorithm

Since the initial success of deep $Q$-learning improvements have been introduced for more stable and reliable training. Listed are a few of the most prominent extensions.

Double DQN corrects a bias for $Q$-value functions to overestimate the action-values [3]. The Double DQN target decouples action selection from evaluation by using the online network to choose the action in the target update:

$$y^{DDQN} = r + \gamma \; Q(s', \arg\max_{a'} Q(s', a';\; \theta);\; \theta^-)$$

$Q$-value functions estimate the value for each state-action pair, which requires a lot of training data. However, in many states the action may not matter. The Dueling network [4] splits the $Q$-value network in into a state value component $V(s)$ and an advantage component $A(s, a)$.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha)$$

Here $\alpha$ are the parameters for the advantage head and $\beta$ those for the value head of a neural network with parameters $\theta$. This split allows for faster learning of

the value function.

Rewards are propagated one step at a time while training with the conventional $Q$-Learning target. The multi-step update rule improves learning efficiency by propagating rewards faster. This is especially helpful in settings with sparse rewards [5]. This monte-carlo return requires the full rollout and not just single transitions.

$$y_t^{MC} = R_t$$

The $n$-step target provides a middle ground between the usual bootstrap target and the multi-step monte-carlo target.

$$y_t^{NMC} = R_{t:t+n} + \gamma^n \; max_{a'} Q(s_t, a'; \theta^-)$$

In the DQN algorithm, samples are drawn uniformly at random from the replay buffer. However not all samples are equally important. Prioritized experience replay [6] samples with a probability according to their TD-error.

### 2.2.2 Policy Based Algorithms

Deep $Q$-value networks have success in large, continuous visual state spaces, but the action space still needs to be discrete. Otherwise choosing the action with maximum $Q$-value would be impossible. Policy based methods and Actor-Critic methods work also in the continuous action spaces arising in control tasks such as robot motion.

Policy based algorithms parametrize the policy $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A} : \pi_\theta(s)$. The policy is optimized directly to maximize the expected reward $E_\pi[R_t]$ using a policy gradient. A commonly used algorithm is the Asynchronous Advantage Actor Critic Algorithm (A3C) [7]. It uses the policy gradient:

$$\frac{\delta \, \mathbb{E}[R_t]}{\delta(\theta)} = \mathbb{E}\left[\frac{\delta}{\delta\theta} \log \pi(a_t \mid s_t) \; (Q^\pi(s_t, a_t) - V^\pi(s_t))\right]$$

Then the gradient scale factor $(Q^\pi(s_t, a_t) - V^\pi(s_t))$ is equal to the advantage $A(s_t, a_t)$ of action $a_t$ in state $s_t$. The value function $V^\pi$ is approximated by an additional neural network, the so-called critic, using the $n$-step update rule. The $Q$-value is estimated by the return $R_t$ of a rollout. A3C uses many instances of agents in parallel to accelerate and stabilize learning. Each parallel learner uses the on-policy samples it generates to update a central policy network.

## 2.3 The Exploration-Exploitation Trade-off

An agent has to decide at every time step. Either make the best decision according to the current estimate or explore and take actions not considered optimal. Exploring can lead to new information about the environment, perhaps reach states not reached before and discover new options which are better in the long run. For large state-action spaces it is not possible to explore the space fully or try every policy possible with an agent, requiring exploitation of current knowledge. On the other hand, a full exploitation strategy would miss out on possible rewards and converge to a local optimum. So, there is a trade-off between the two and they need to be balanced.

Finding an optimal policy in DRL requires efficient methods to both explore the environment, as well as exploit the current knowledge about it. Complex environments introduce new challenges to both exploration and exploitation. In the following, a diverse set of approaches is presented aiming to overcome these challenges.

Chapter 3

# Methods for Exploration

A good exploration method should explore promising regions and help an agent gather useful information about the environment. Many different methods are used to guide exploration. They follow one of a few principles.

The first principle is random exploration. This is the easiest principle and requires no extra information about an environment. An example is the $\epsilon$-greedy strategy as used in the DQN algorithm. Another one is the softmax strategy: It randomly chooses an action weighted according to the softmax distribution over the estimated expected return for each action. Both strategies are limited as they stray away from the current policy one step at a time. These are considered shallow exploration, failing to discover strategies which require few steps from the current policy. The $\epsilon$-greedy strategy is also unguided and will make the same mistakes repeatedly.

A second principle is the optimism in the face of uncertainty principle. These methods require a measure for the uncertainty of the value alongside the estimation of the value itself. The principle says that the more uncertain an agent is about an action, the more important it is to explore. One example is the upper confidence bound (UCB) strategy. It estimates the an upper bound on the value which holds with high probability. Choosing the action with the highest upper bound yields either the best action or reduces uncertainty over it, thus lowering the upper bound. See Figure 3.1 for a visualization of choosing an action according to the UCB principle. Similarly, Thompson sampling keeps a distribution of the value uncertainty. Then the goal is to choose a policy according to the probability that it is optimal [8]. Both these methods keep track of uncertainty in their estimation and reduce it in a guided manner.

The third principle is the information state search principle. The goal in exploration is to gain information. Not just information about potential rewards, but also the dynamics of an environment or other general aspects of it. Estimating

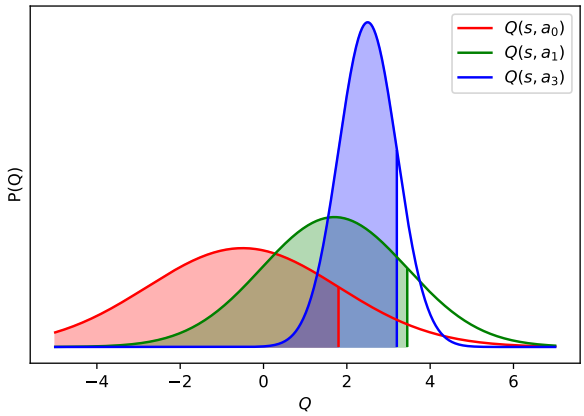


Figure 3.1: The uncertainty when choosing action $a_0$, $a_1$ or $a_2$ in some state $s$ given by a probability distribution. Shaded is the confidence interval containing the correct value with high likelihood. The UCB principle states the best action for exploration is $a_1$ as it has the highest limit to the confidence interval.

this gain is done by estimating information an agent has about its environment in an information state. The value of this information is then weighted against the value of the reward leading to an explicit trade-off between exploration and exploitation.

Exploration strategies also differentiate themselves in the space they explore. First there is state-action exploration. These use knowledge about the state-action space. For example, by choosing a different action each time a state is visited to guide exploration. Secondly there is parameter space exploration which alters the parameters of a policy to achieve exploration behavior.

The following examples for exploration strategies in DRL each follow one or more principles to achieve efficient exploration.

## 3.1 Intrinsic Motivation based Exploration

Intrinsic motivation is a psychological drive that guides behavior even in the absence of external rewards. The motivation is based on internal desires such as curiosity about an environment. Consider for example a robot with the task to stack blocks. It will only receive an extrinsic reward when the blocks are all on one stack. Randomly choosing the correct sequence of actions is highly improbable, but with intrinsic motivation the robot is curious about what it can do with blocks and thus guided towards solving the task.

To implement an intrinsically motivated agent the reward it receives is augmented with the intrinsic reward $r^i(s, a)$ scaled by some factor $\beta$.

$$\tilde{r}(s, a) = r(s, a) + \beta r^i(s, a)$$

The hyperparameter $\beta$ has to be tuned. A high $\beta$ encourages exploration, while the smaller it is, the more focus lies on the original objective. Modifying the reward means that these methods can be used with any reward-based reinforcement learning algorithm.

In general, the intrinsic rewards are based on a forward dynamics model learned concurrently with the policy. This way an agent learns the consequences of its own actions. The agent is driven towards regions where predictions are less accurate and as a result improves its forward prediction model in these regions.

A classic forward dynamics model predicts the next state $s_{t+1}$ given the current state and actions $(s_t, a_t)$, but as states are very high dimensional the model is learned in a feature space, e.g. computed by an auto-encoder (Stadie et al, [9]). Building a state representation that only encodes part of an environment the agent can influence improves robustness of the exploration strategy in the presence of distracting objects (Pathak et al. [10]; Haber et al. [11]). On top of implementing a curiosity drive from a dynamics model, also a homeostatic drive can be implemented, which aims to keep states familiar such that the agent acts according to familiar patterns (de Abril & Kanai, [12]). These methods use the error of a forward dynamics model as the intrinsic reward. This model is a deterministic function, so the error is only informative in deterministic environments. Other methods model the full MDP with stochastic transitions and use the KL-divergence between true transitions probabilities and a learned model (Achiam & Sastry, [11]). Instead of using the error of these dynamics models the intrinsic reward can also be based on parametric uncertainty (Houthooft et al. [13]). The previous intrinsic motivation methods all alter the reward an agent receives to guide its behavior. But intrinsic motivation can also be useful for learning interesting aspects of an environment. These methods aim to improve state representations through auxiliary tasks (Jadenberg et al. [14], Shelhamer et al. [15]).

We now explore two examples in more detail.

### 3.1.1 Curiosity-driven Exploration by Self-supervised Prediction

Pathak et al. [10] computes the intrinsic reward based on the prediction error in a forward dynamics model. Since for high dimensional inputs (such as images), predicting the next input is challenging, there is a need to compute a simpler

state representation. Also using the state space directly would mean that failing to predict movement of background objects leads to a high intrinsic reward, even though these changes do not affect the performance of an agent.

These issues are addressed with the Intrinsic Curiosity Module (ICM) (see Figure 3.2). The intrinsic reward is the error of the forward model $f$ computed in feature space. $r_t^i = \beta \| f(\phi(s_t), a_t) - \phi(s_{t+1}) \|^2$. Computing the error in feature space allows this method to scale to complex inputs. The feature transformation $\phi$ is trained using a self-supervised inverse dynamics model. The goal of the inverse dynamics model is to predict the action $a_t$ from a transition $s_t$ to $s_{t+1}$. Consequently, there is no incentive for features $\phi$ to encode parts of the input that can't be affected by the agent.

The authors experiments show how an agent using these intrinsic rewards can learn to play a level of Mario even in the absence of extrinsic rewards. Nevertheless, basing the intrinsic reward on a forward model has limitations. Environments must be deterministic. If actions can have random consequences, the next state could not be accurately predicted leading to high intrinsic reward that can't be reduced over the training.

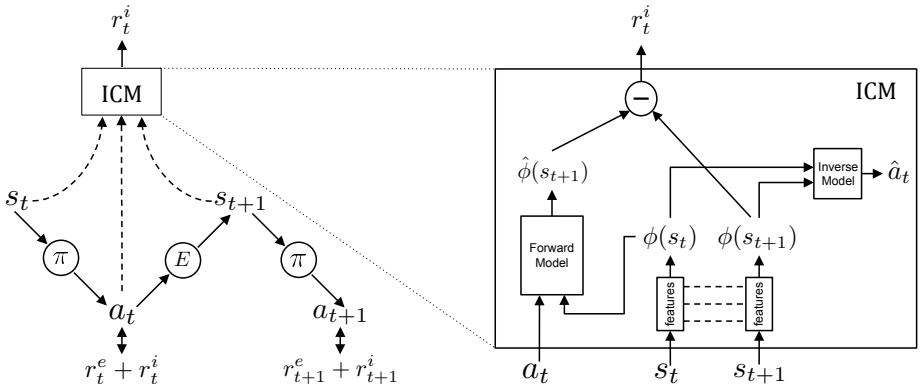


Figure 3.2: The Intrinsic Curiosity Module (ICM) [10] used to compute the intrinsic reward $r_t^i$. It consists of an inverse dynamics model learning a feature transformation $\phi$, which only captures features the agent can affect. Then the forward dynamics model predicts the next state in this feature space and the prediction error is taken as an intrinsic reward.

### 3.1.2 Unsupervised Auxiliary Tasks

In DRL state spaces can be complex. Images from video for example, contain of a vast range of signals and finding features relevant to an agent's policy is challenging in the absence of rewards. The UNsupervised REinforcement and

Auxiliary Learning (UNREAL) algorithm (Jaderberg et al. [14]) finds good state representations by training on auxiliary tasks maximizing other intrinsic rewards, rather than the extrinsic one from the environment.

Changes in intensities often relate to important changes in the environment and the agent learns to change them maximally. For each auxiliary task $c$ a separate agent is trained with its own value function $Q^c$. These share the first layers of a neural network with the original A3C agent approximating $V$ and $\pi$. As a result, the shared network computes features that are relevant to the agent. Data efficiency of the base A3C agent is further improved by two additional techniques called reward prediction and value replay. Reward prediction is another auxiliary task and again uses the same base layers, but a simpler overall architecture to predict the immediate reward given some state. Similarly to DQN, value function replay uses the replay buffer to update the value function $V$ using a multi-step update rule. The full loss optimized by the UNREAL agent is then:

$$L_{UNREAL} = L_{A3C} + \lambda_{VR} L_{VR} + \lambda_{PC} \sum_c L_Q^{(c)} + \lambda_{RP} L_{RP}$$

Here $\lambda_{VR}, \lambda_{PC}$ and $\lambda_{RP}$ balance the value replay, pixel-control and reward-prediction losses respectively.

Augmenting A3C as a baseline with these auxiliary tasks results in a faster and more robust learning algorithm, as is evidenced by the authors Atari benchmark results. In contrast to other intrinsic motivation methods, UNREAL does not require learning a dynamics model, but only learns to control certain aspects of the input. However not all pixels can be controlled. Some may be random, some may be irrelevant to the task of an agent. It is unclear how training to control these affects the agent.

## 3.2 Count-Based Exploration

Count-Based Exploration follows the optimism in the face of uncertainty principle. These value-based algorithms count state(-action) visits: A small visit count gives a high exploration bonus as the agent is uncertain about the value. During training the visit count gets higher, the agent develops a better value approximation and the uncertainty decreases. For a finite MDP and tabular RL, the *Model-Based Interval Estimation with Exploration Bonus* (MBIE-EB) algorithm (Strehl & Littman [16]) offers strong theoretical guarantees. It is model-based and thus estimates the transition probabilities $\hat{P}$ and the stochastic reward $\hat{r}$. The method counts state-action visits $n(s, a)$ and solves a modified

Bellman optimality equation:

$$\tilde{Q}(s,a) = \hat{r}(s,a) + \frac{\beta}{\sqrt{n(s,a)}} + \gamma \sum_{s'} \hat{P}(s' \mid s,a) \max_{a'} \tilde{Q}(s,a)$$

MBIE-EB is PAC-MDP (Probably Approximately Correct in Markov Decision Processes), meaning in can find an approximately correct value table with high probability in a time polynomial in the sizes $|\mathcal{S}|, |\mathcal{A}|$ and the approximation error. MBIE-EB inspires the bonus used in count-based methods for DRL. However applying these bonuses directly to DRL with large to infinite state spaces is impossible, as most states are never visited and consequently the bonus is the same for most states.

Count-based methods are also intrinsic motivation based, but in contrast to the previous methods these do not model the environmental dynamics. They solely rely on the state space to guide exploration toward novel states. As they also add an intrinsic reward they share the common downside of changing an agent's objective. In this setting an agent can visit a less frequently visited state many times, even though the value in this state is already estimated.

For large state spaces several for count-based exploration have been proposed. These use pseudo-counts based on a visit-density model over states (Bellemare et al. [17]). The visit-density model predicts the probability of visiting some state given the history of state visitations. Ostrovski et al. [5] explore the importance of a good density model and use a strong neural network based model which is learned alongside the value function. Computing pseudo-counts based on a locally-similar hash function and then counting visits in each bucket has also been successful (Tang et al. [18]). In all these approaches the similarity measure between states plays a critical role. Novel states should have low probability in the density model, while states with no important differences to an agent should have high probability or map to the same bucket given some hash function. These pseudo-counts are based on state representations distinct from the one used in the Q-value network. Martin et al. [19] compute the pseudo-count directly in the feature space of a linear Q-value approximator, ensuring a task-relevant density model is produced. Machando et al. [20] develop a stochastic successor representation, which implicitly counts feature visitations and thus leads to a count based intrinsic reward without a density model.

### 3.2.1 Unifying Count-Based Exploration and Intrinsic Motivation

One way to remedy-count based approaches is to use a density model (Bellemare et al. [17]). The density model $\rho : \mathcal{S} \to \mathbb{R} : \rho(s)$ predicts the probability of

visiting a state following some policy. A new unvisited state which is similar to the already seen states can have a positive probability, compared to the visit count which would be zero for all new states. Given a density model one can derive pseudo counts.

With $n$ recorded states $s_{1:n}$ the probability of state $s$ is given by: $\rho_n(s) = \rho(s; s_{1:n})$. The recording probability $\rho'_n(s) = \rho(s; s_{1:n,s})$ is the probability of observing state $s$ again after training on $s$ additionally. Then the pseudo count $\hat{N}_n(s)$ should increase by 1, by a single observation. Let $\hat{n}$ be the pseudo-count total.

$$\rho_n(s) = \frac{\hat{N}_n(s)}{\hat{n}}, \;\; \rho'_n(s) = \frac{\hat{N}_n(s) + 1}{\hat{n} + 1}$$

Solving this linear system yields the pseudo count $\hat{N}_n(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s)-\rho_n(s)}$. Here $\hat{n}$ is the pseudo count total. An intrinsic reward is then defined:

$$r^i(s) = (\hat{N}(s) + \epsilon)^{-\frac{1}{2}}$$

This reward is inspired by the optimism in the face of uncertainty principle and $\epsilon$ is a small number for numerical stability.

The density model $\rho$ has to be designed specifically for each state space. Ideally the density model generalizes over differences that are not important for an agent (like background objects), while distinguishing between key features in the state space. To give meaningful pseudo counts it has to be learning positive ($\rho'_n(s) > \rho_n(s)$) and each observed state has to be used exactly once to train the density model. In their experiments a Context Tree Switching (CTS) model [21] predicts the probability of a pixel intensity based on its neighbors. Results show that for games with sparse rewards, performance significantly improves when using intrinsic rewards from pseudo counts.

The pseudo-counts in this method rely on strict assumptions about the density model like learning positiveness and using samples exactly once to train the density model. In Count-based Exploration using Neural Density Models (Ostrovski et al. [5]) it is shown how these assumptions can be relaxed and how a better density model can improve performance. The pseudo count is approximated via a prediction gain $PG_n(s) = \log \rho'_n(s) - \log \rho_n(s)$. Then $\hat{N}_n(s) \approx (e^{PG_n(s)} - 1)^{-1}$. A density model based on neural networks still is not learning positive, but the prediction gain can be capped to positive values. Their experiments show that PixelCNN, an advanced density model, can further improve exploration over the previous CTS based density model.

## 3.3 Ensemble approaches

Ensemble methods consist of a collection of base models which are then combined into one stronger model. In supervised machine learning these techniques are used to improve predictive power like in boosting or to reduce variance of the model like in bagging. Here are approaches using an ensemble of policies to improve exploration over base methods in DRL.

### 3.3.1 Bootstrapped DQN

In Bootstrapped DQN (Osband et al. [22]) build an ensemble of $K$ $Q$-value functions. The individual $Q$-value functions $Q^k$ represent samples of the posterior over estimates for $Q^*$ and reflect the parametric uncertainty in the learning algorithm.

These $K$ $Q$-value functions are implemented as heads to a shared network (Figure 3.3. The shared part allows for an efficiently learned feature representation at the cost of diversity between the heads. To ensure diversity between the heads two techniques are tested: Random initialization of heads and the bootstrap sampling scheme. Bootstrap sampling trains on random subsamples of the training data for each head, but the authors experiments reveal that sharing all transition samples retains enough diversity.

Exploration is inspired by Thomson sampling: At the start of an episode, one of the $K$ $Q$-values $Q^k$ is sampled and then the greedy policy selects actions for the full episode. Taking actions according to the trained $Q^k$ instead of randomly as in $\epsilon$-greedy leads to a directed exploration method. The selected greedy policy is consistent over multiple time steps as it will consistently choose the same action given the same state. Combined this leads to a deep exploration method: It is directed exploration and consistent over multiple time steps. This is necessary in for problems where reward is not immediately following an action.

### 3.3.2 UCB Exploration

UCB Exploration (Chen et al. [23]) uses the same ensemble as bootstrapped DQN. The exploration is different however and guided by the uncertainty between the $Q$-value heads. The approach follows the optimism in the face of uncertainty principle. At each time step the action chosen is the one with the highest upper confidence bound across all $K$ $Q$-value heads. Specifically the action is $a_t \in \arg\max_a(\tilde{\mu}(s_t, a) + \lambda \cdot \tilde{\sigma}(s_t, a))$. Here $\tilde{\mu}(s_t, a)$ is the sample mean and $\tilde{\sigma}(s_t, a)$ is the empirical standard deviation of the ensemble.

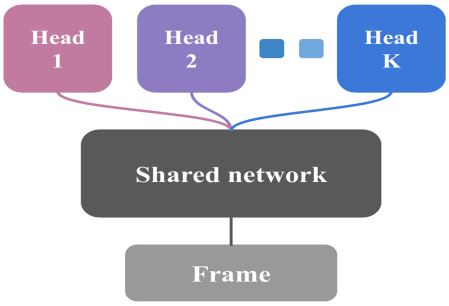


Figure 3.3: The Bootstrap DQN network with $K$ heads [22]. The shared network architecture learns shared features from the input frame for computational efficiency at the cost of diversity between the value heads.

In contrast to bootstrapped exploration the policy head is no longer kept for the full episode. Instead, the policy is similar to the ensemble strategy, which takes the action with most votes at each time step and therefore has a high $Q$-value. Further, taking actions with high standard deviation reduces uncertainty at difficult decision points. In comparison this leads to a stronger policy in a majority of Atari games over the Double DQN and Bootstrapped DQN .

## 3.4 Parameter Space Exploration

Aside from the ensemble strategies parameter exploration can also be achieved by directly introducing noise to the parameters of a policy. In deep reinforcement learning the policy is given by a neural network with trained weights. Changing a single weight in the policy network can lead to a complex change in policy that is state dependent and consistent over multiple time steps.

### 3.4.1 Parameter Space Noise

Parameter Space Noise for Exploration perturbs the weights with additive gaussian noise (Plappert et al. [24]). For a policy network with current parameters $\theta$ the new perturbed version is given by: $\tilde{\theta} = \theta + N(0, \sigma^2 I)$. The amount of noise needs to be adjusted carefully to get a meaningful policy. Deep neural networks are more sensitive to noise introduced in earlier layers. Layer normalization [25] is applied between perturbed layers and consequently, the same noise scale can be applied across all layers. The amount of noise $\sigma$ is adaptively chosen to keep the distance between the unperturbed and perturbed policy $d(\pi_\theta, \pi_{\tilde{\theta}}) \leq \delta$ small. This distance $\delta$ is annealed during training to trade off exploration with exploitation. The distance $d$ between policies is implemented differently for each learning algorithm. For deep $Q$-learning the probabilistic softmax policy $\pi_\theta(a|s) = \frac{expQ_\theta(s,a)}{\sum_{a' \in \mathcal{A}} expQ_\theta(s,a')}$ is used. Then the difference is given

by the KL-divergence $D_{KL}(\pi_\theta||\pi_{\tilde{\theta}})$ estimated on the states traversed in the replay buffer.

The exploration strategy is then to sample new parameters at the start of the episode. Like bootstrapped DQN this results in a deep exploration strategy, but with possibly more diverse policies as they are sampled from a continuous distribution of policies. Figure 3.4 shows how parameter space noise compares to bootstrapped DQN on a test task and how parameter space exploration can find policies that $\epsilon$-greedy does not find.

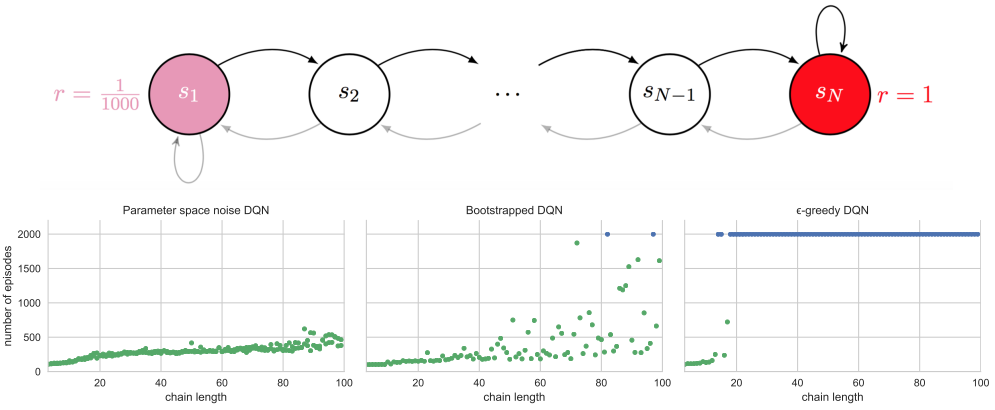


Figure 3.4: Chain experiment [24]. The optimal policy is to go right in every state and receive the big reward in the rightmost state. The graphs show the median number of episodes until the environment is considered solved. In this environment parameter space exploration is able to effectively learn, while $\epsilon$-greedy exploration requires data exponential in the chain length.

### 3.4.2 Noisy Networks

Another approach to parameter space exploration is to use noisy networks (Fortunato et al. [26]). A noisy network is a neural network with a random distribution over its weights $\theta$. These weights are given by $\theta = \mu + \Sigma \odot \epsilon$, where $\mu$ contains the mean for each weight and $\Sigma$ the noise scale. $\epsilon$ is randomly distributed according to some zero mean noise distribution and $\odot$ is the element-wise product. Then $\zeta = (\mu, \Sigma)$ is the new set of learnable parameters. The loss $L(\theta)$ of a network is then replaced with the expectation over the noise $\epsilon$. So the new loss is $\bar{L}(\zeta) = \mathbb{E}[L(\theta)]$ which is optimized with stochastic gradient descent like normal deep $Q$-learning or other learning algorithms. A noisy linear layer is depicted in Figure 3.5. A neural network consisting of noisy layers has twice the number of parameters and can be trained in DRL algorithms with only minor adaptations. In their experiments, the authors adapt DQN, DDQN with

Dueling and A3C. For all they show improvement on Atari benchmarks without significantly increasing computational cost. In the Rainbow algorithm (Hessel et al. [27]), the noisy network is one of many extensions used to improve DQN.

Noisy networks give rise to parameter space exploration, since a new policy or value network can be sampled at every time step. Having the noise level learnable means, it is automatically adjusted during training, in contrast to the parameter space noise approach presented before. The noise level is problem specific and automatically learned.

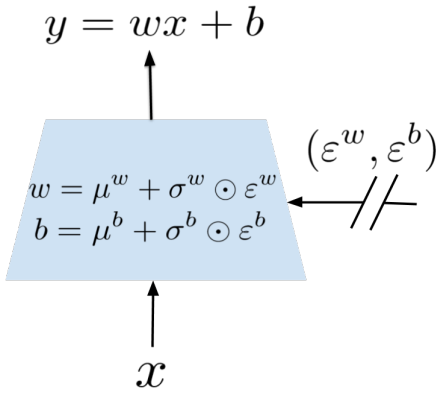


Figure 3.5: A noisy linear layer. The weight matrix $w$ is computed as $w = \mu^w + \sigma^w \odot \epsilon^w$ and analogously the bias $b$. While the noise $\epsilon^w, \epsilon^b$ is sampled at random, the parameters $\mu^w, b^w, \sigma^w, \sigma^b$ are learnable.

## 3.5 Distributed Architectures

A general idea in deep learning is to use more computational resources and work with larger datasets. In Reinforcement Learning interactions with the environment are sequential by nature, however multiple approaches exist to efficiently use parallel or distributed compute architectures. The idea is to decouple acting and learning. Actors interact with their own environments to collect experiences, while a learner's job is to update the policy based on the collected experiences. Aside from utilizing computational recourses a distributional approach has advantages to exploration. Using slightly different policies, e.g., induced by changing $\epsilon$ in the $\epsilon$-greedy exploration policy, diversifies the set of collected experiences. Even without explicit manipulation, policies in a parallel and asynchronous setting differ depending on the learners' synchronization and thus likely explore different states. The diverse experiences help break correlations between data samples and thus improve learning stability. Exploration is also aided since multiple learners are able to cover a larger amount of the state space.

In the General Reinforcement Learning Architecture (Gorila) (Nair et al. [28]) multiple actors and multiple learners are distributed across several machines. The learners send gradient updates to a global parameter server and the actors send their experiences to a replay server. The Distributed Prioritized Experience Replay (Ape-X) (Horgan et al. [29]) streamlines the architecture (See Figure 3.6) using a single GPU based learner and by communicating through collected transitions, rather than gradients to reduce network traffic. Importantly the learners generate vast amounts of data and then select the most useful events using prioritized replay.

Actor-Critic methods have also benefited from parallel architectures, despite being on-policy algorithms in general. In on-policy algorithms experiences have to be generated by the same policy as is updated. A3C (Mnih et al. [7]) uses multiple actor-learner threads updating a global network asynchronously. The GPU Asynchronous Actor-Critic (GA3C) (Babaeizadeh et al. [30]) and Parallel Advantage Actor-Critic (PAAC) (Clemente et al. [31]) utilize the GPU for efficient training of actor-critic methods. Gruslys et al. [32] decouple acting and learning in the off-policy Reactor (Retrace-Actor) agent based on the retrace algorithm [33]. Reactor also improves sample efficiency by incorporating $n$-step updates and new form prioritized replay. The importance weighted Actor-Learner Architecture (IMPALA) (Espeholt et al. [34]) uses GPU based learners and scales a large number of actors. It is the base algorithm for many recent state-of-the-art algorithms.

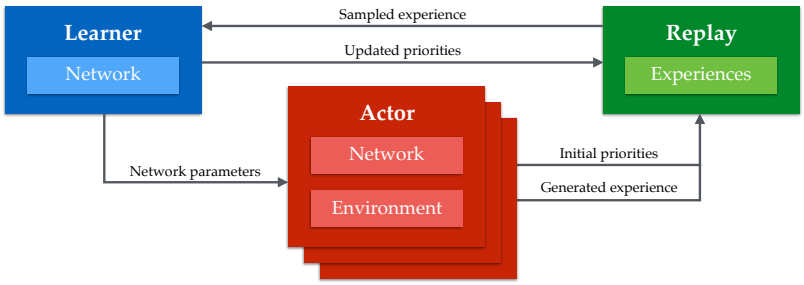


Figure 3.6: The Ape-X architecture [29]. A single learner updates the network parameters and the learning error is sent to update priorities of the replay samples. The actors receive their policy weights from the learner and store their experiences in the replay buffer. This architecture does not require gradient updates to be sent, reducing network traffic and allowing for more actors.

## 3.6 Curriculum Learning

Human learning is based on an organized school curriculum. Simple concepts are introduced first, followed by concepts of increasing complexity that build upon the earlier ones. This curriculum eases learning of new abstractions and can increase training speed. A curriculum can also be beneficial as a learning strategy for machine learning. Here the idea is to first learn simple tasks and gradually increase complexity. These tasks are for example given by increasingly difficult samples for a classifier or regressor (Bengio et al. [35]).

Curriculum learning is also applicable in DRL. A robotics task may be specified with a single reward for completing the objective. Reaching the rewarding state is a too complex task to start with. So instead of learning a policy to directly reach this state, one can learn a universal policy $\pi(a|s, g)$ (Schaul et al. [36]). It takes a goal state $g$ as a parameter and tries to reach it with the fewest number of transitions possible. Choosing the goals according to a curriculum of solvable tasks, it is possible to train an agent with increasingly complex tasks. In the end this produces a policy that is able to reach any state from any other, including the original objective. Curriculum learning bares similarity to multi-task learning. A robot may be required to solve many tasks within the same environment and curriculum-based approaches offer a guided way to learn these tasks.

Asymmetric Self-Play (Sukhbaatar et al. [37]) trains two RL agents. A teacher (Alice) to generate tasks specified by a goal state, and a student called Bob to learn and complete them. Bob's reward is determined by how fast he reaches the target state. At the end of an episode Bob receives $r_B = -t_B$ if he is successful or $t_{Max} - t_A$ otherwise. Alice's reward is $r_A = \max(0, t_B - t_A)$. Here $t_A$ is the number of time steps it takes Alice to set up a task by executing her policy and reaching $g$. For a maximal reward, Alice presents the simplest task Bob cannot complete. Then the inter-play between the agents leads to an automatically generated curriculum.

Training Alice's policy simply for generating goals is inefficient as it requires separate interactions in the environment. Furthermore the self-play approach has a biased exploration towards a small region of space where Alice and Bob get stuck [38]. Florensa et al. [39] use a generative adversarial neural network (GAN) [40] to address the problem. The GAN takes the role of the teacher. It consists of goal discriminator judging if a goal $g$ is at an appropriate difficulty and a goal generator producing said goals. Another way to find goals is presented in Hindsight Experience Replay (Andrychowicz et al. [41]). It looks at the collected trajectories $(s_0, ..., s_T)$ from the replay buffer. If the real goal state is not reached, it sets a new goal $g = s_T$ and learns how it can be reached. This

way it is possible to balance rewarding trajectories with non-rewarding ones in the replay buffer for an improved learning efficiency.

Instead of setting tasks by giving a goal state, it is also possible to specify a task with a start state. Reverse Curriculum Learning (Florensa et al. [38]) builds a curriculum of start states which are increasingly far away from the set of goal states. As a result, the agent learns to reach its goal from a large set of start states, which improves robustness over an agent starting from a single state. This approach requires some additional assumptions about the environment. For one it has to resettable to any state in order to start an episode. And at least one goal state $g$ must be provided. Furthermore for any states $s_1, s_2 \in \mathcal{S}$ there has to be a trajectory from $s_1$ to $s_2$. New start states are sampled using a random policy starting from the current start state, in the reverse direction the agent has to traverse. Another approach with similar requirements is based on region growing (Molchanov et al. [42]). It keeps track of a reachability region consisting of the states where the policy is already capable of moving from any state to any other. This region is gradually extended during training until a universal policy is learned for the full state space.

Rather than building a curriculum of increasingly difficult tasks, Czarnecki et al. [43] build a curriculum of increasingly complex agents. The approach might start with a policy on a few actions and gradually change to a policy over the full action space, thus learning in environments with big action spaces more efficiently.

## 3.7 Hierarchical Learning

An enduring problem in DRL is credit assignment over long periods of time. In Atari games with sparse rewards it could be hundreds of frames between reward signals. The standard way to address this is frame stacking. Here multiple (typically 4) frames are used as state input, reducing time steps between reward signals. Hierarchical reinforcement learning offers a more principled approach by introducing RL models at multiple time scales. Typically a high-level policy choosing subgoals or options and multiple low-level policies executing primitive actions to achieve these subgoals. The high-level policy then operates at a lower temporal resolution, enabling faster learning.

The options framework (Sutton et al. [44]) introduces Markovian options $\omega \in \Omega$ to formalize this hierarchy. An option $\omega$ is a triplet $\langle I_\omega, \pi_\omega, \beta_\omega \rangle$. Here $I_\omega \subset \mathcal{S}$ is an initiation set (typically $I_\omega = \mathcal{S}$), $\pi_\omega$ it an inter-option policy and $\beta_\omega : \mathcal{S} \to [True, False]$ is a termination function. The set of options $\Omega$ leads

to higher level RL problem, where the meta-policy $\pi_\Omega(s)$ learns which option $\omega$ to choose given some state $s$. This meta policy should then be more efficient at exploring the state space, since it can take bigger steps.

Implementations in DRL for hierarchical learning differ in the options considered and their terminations functions. Hierarchical DQN (Kulkarni et al. [45]) implements DQN agents for two levels of hierarchy with a controller and meta-controller (see Figure 3.7). Their set of goals (options) $\Omega$ is designed for each environment and specified in advance. It might consist of the entities in a goal world. Then the termination function decides if the low-level agent is able to navigate close enough to this target entity.

Ideally a DRL agent would decompose tasks into options on its own and find abstract goals for the low-level agent. The option-critic architecture (Bacon et al. [46]) offers a way to learn both options and termination functions automatically. Thus building an end-to-end algorithm for general DRL problems. However the learned policy often degenerates to one of two solutions: Either only one option is active, implementing the whole policy or the meta-policy chooses another option for every time step. Feudal Networks (Vezhnevets et al. [47]) address this issue by separating learning between meta-policy and inter-option policies and also by generating explicit subgoals in a latent space. Harb et al [48] address frequent termination in the option-critic architecture with deliberation costs for terminating options.

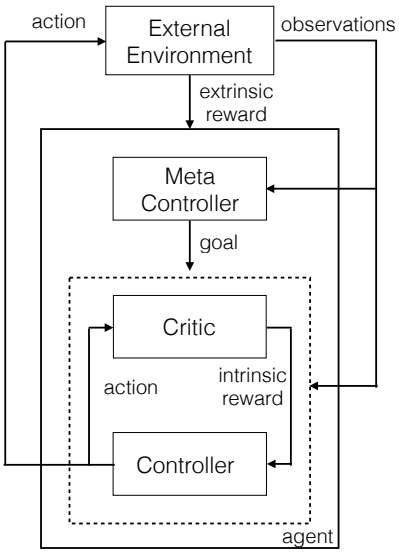


Figure 3.7: The perception-action-learning loop for hierarchical DQN [45]. The agent consists of a meta controller and controller operating at two different time scales. The meta controller has as his action space a discrete set of subgoals and learns to choose those to maximize extrinsic reward. The low level controller selects primitive actions in the environment in order to complete these subgoals. The critic serves as a termination function, deciding when a goal is achieved and awarding an intrinsic reward.

Chapter 4

# Methods for Exploitation

Exploration methods are not only helpful in hard exploration problems but can also improve sample efficiency in general. Another reason for learning algorithms to be inefficient besides poor exploration strategies is poor learning efficiency or exploitation. The methods presented hereafter do not improve an exploration strategy, but rather exploit their policy or a policy demonstrated to them. They are capable of solving problems where $\epsilon$-greedy exploration was thought to be insufficient [49].

## 4.1 Episodic Control

Episodic memory is a type of human memory besides procedural and semantic memory [50]. Procedural memory stores procedures like walking or riding a bicycle and is linked to model-free reinforcement learning. Semantic memory stores general world knowledge like ideas or concepts and is linked to model-based reinforcement learning. In contrast episodic memory stores specific experiences and events like visiting a birthday party. These memories lead to episodic learning, which enables the brain to learn from a single experience rather than through repeated trial or from related concepts. Episodic control is inspired by these mechanisms for rapid learning in the human brain.

Current DRL algorithms can take millions of interactions with an environment. They may experience a highly rewarding sequence of events but fail to capitalize on it. Since DRL algorithms rely on neural network based value approximations they have several issues: Stochastic gradient descent in neural networks requires small step sizes, since too big steps risk catastrophic interference. Another cause for slow learning is the bootstrap update rule. Deep Q-learning propagates rewards only one step at a time and consequently needs to experience the same rewarding event multiple times. The target network introduced to improve stability also changes only periodically, further slowing down training.

### 4.1.1 Model-free episodic control

Model-free Episodic Control (Blundell et al. [49]) implements a simple model of the hippocampal episodic control. General DRL algorithm assume a stochastic environment which causes further inefficiencies, since the expectation over a random process has to estimated. Model-free episodic control assumes a deterministic environment.

Episodic memory im modeled with a table $Q^{EC}(s, a)$ storing the most rewarding return for each state-action pair. At the end of each episode it is updated as follows:

$$Q^{EC}(s_t, a_t) = \begin{cases} R_t & \text{, if } (s_t, a_t) \notin Q^{EC} \\ \max\left(Q^{EC}(s_t, a_t), R_t\right) & \text{, otherwise} \end{cases}$$

This learning update is not valid in stochastic environments as it would lead to a risky policy that takes an action even if the outcome of this action is very unlikely. However as desired, the policy implicated by $Q^{EC}$ will reproduce the most successful sequence of events even if they were experienced only once.

A DRL algorithm needs to generalize to states not visited and thus not in the table $Q^{EC}$. Hence the $Q$-value at some state-action pair $(s, a)$ is estimated by using values from the nearest neighbors $s^{(1)}, ..., s^{(k)}$:

$$\hat{Q}^{EC}(s_t, a_t) = \begin{cases} \frac{1}{k} \sum_{i=1}^{k} Q^{EC}(s^{(i)}, a) & \text{, if } (s, a) \notin Q^{EC} \\ Q^{EC}(s, a) & \text{, otherwise} \end{cases}$$

The model-free episodic control algorithm will then use the estimate $\hat{Q}^{EC}$ for policy rollout and update its table $Q^{EC}$ according to the observed sequence of states in an episode.

Episodic Control also works on transformed states $s_t = \phi(o_t)$ rather than the direct observation $o_t$. This transformation not only reduces the memory footprint but choosing the right transformation $\phi$ can improve generalization of the algorithm. A good transformation should ignore irrelevant details and distances in the feature space should correlate with differences in the action-value. The authors test two transformations: Random projections of observations into a lower dimensional subspace and a transformation learned by a Variational Autoencoder (VAE) [51] trained on state observations collected with a random policy. Their experiments on Atari games show how $\epsilon$-greedy exploration can lead to much faster policy improvement when combined with an algorithm capable of quickly learning from newly discovered strategies. But while initial performance is greatly improved, policies represented with neural networks generalize better once later stages in a game can be reached.

### 4.1.2 Neural Episodic Control

Neural Episodic Control (Pritzel et al. [52]) also utilizes Q-value function approximations based on an episodic control rather than neural networks. It addresses the same issues with neural network based value approximations, but also aims to capitalize on the benefits of them. Neural networks are able to learn feature transformations from high dimensional inputs.

Neural Episodic Control builds a fully differentiable neural dictionary to store the returns for each transformed state. The feature transformation is a neural network. Combined with a lookup in the neural dictionary this leads to a differentiable architecture which can learn abstract state representation much like conventional DRL architectures. See Figure 4.1 for a schematic of the full architecture. Crucially the learned representation focuses on aspects that are helpful to predict rewards, whereas handcrafted features, random projections or feature mappings based on a VAE ignore reward. So irrelevant details should be ignored and distance in the feature space should be more meaningful.

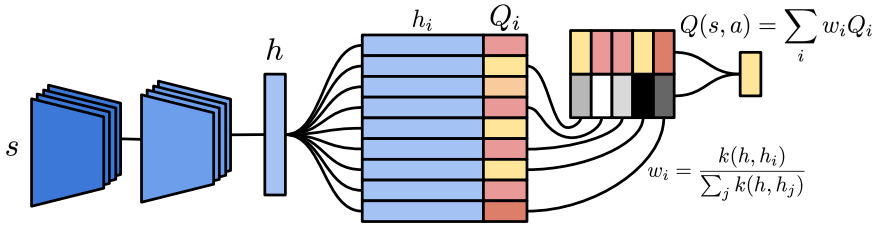


Figure 4.1: The full Neural Episodic Control architecture for a single action $a$ [52]. The input state $s$ is transformed with convolutional layers to a representation $h$. The dictionary stores all representations $h_i$ encountered during training as well as their $n$-step $Q$-value $Q_i$. To estimate the Q-value at a new state $s$ with representation $h$ the nearest neighbors are found in the dictionary and their contribution to $Q(s,a)$ is weighted according to some kernel distance function $k$. This whole architecture is differentiable, so the error between $Q(s,a)$ and the $n$-step target can be backpropagated to update weights in the convolutional layers.

## 4.2 Learning from Demonstrations

The previous methods all assumed the familiar RL setting of an agent learning in an environment solely based on rewards. But in real world settings more information may be available. A self-driving car could utilize data captured by real drivers. Some problems such as recommendation systems may already have a working system based on previous RL agents or other techniques. In

these settings the goal of a reinforcement learning algorithm is to imitate the previous policy and ultimately to improve on it. In settings with sparse rewards, demonstrations can replace a good exploration strategy and learn policies by exploiting the demonstrated one. These methods typically also increase learning speed as they learn from the given demonstration or previous systems.

Pure imitation learning finds a policy by imitating an expert. These methods do not require rewards to specify a task as in RL. Assessment of a policy is solely based on how close it matches the demonstrated policy. Imitation learning can be split into two categories: Those working with a passive collection of demonstrations and those with an active expert. In the passive collection setting the demonstrations are given before training starts. A common method is behavioral cloning which minimizes a policy loss between the demonstrated policy and learned one $\pi_\theta$ parameterized by $\theta$. For a deterministic policy with discrete actions and a collection of $N_D$ demonstrations $(s_i, a_i)$ the loss is given by:

$$L^{BC} = \sum_{i=1}^{N_D} \|\pi_\phi(s_i) - a_i\|^2$$

This loss can be minimized by modelling $\pi_\theta$ as a classifier and using standard supervised learning.

Behavioral cloning methods have for example been used for autonomous driving (Bojarski et al. [53]). State-of-the-art imitation learning methods include (Ho et al. [54]; Ho & Ermhon [55]). However the resulting policy will unlikely work outside the manifold of the state space spanned by the demonstrations. Furthermore the i.i.d. assumptions made in supervised learning is violated. In the active learning from demonstrations setting an expert is also available during training and policies can be tested in an environment. An active learning strategy can help overcome the shortcomings in the passive learning setting (Ross et al. [56]; Sun et al. [57]). But pure imitation learning is still limited, as the learned policy can never improve upon the one demonstrated by the expert.

Combining DRL and imitation learning offers more flexibility in the learned policies. Policies are evaluated according to the RL return. The passive learning from expert demonstrations again assumes a collection of demonstrations is available before training begins. Human checkpoint replay (HCR) samples checkpoints from human demonstrations to start an episode (Hosu & Rebedea [58]). Here the demonstrations help an agent reach states that a DRL agent may have never explored on its own. Deep Q-Learning from Demonstrations (Hester et al. [59]) adapts the behavioral loss from pure imitation learning for DRL. The demonstrations are also kept in the replay buffer to sample from, instead of

only relying on its own experiences. The same replay strategy is also applied to off-policy DRL for continuous control tasks (Vecerik et al. [60]). Nair et al. [61] extend this approach with a behavioral cloning loss, as well as Q-filtering, which automatically ignores expert demonstrations as they become obsolete. Learning from demonstrations can also be combined with exploration algorithms to help initial learning performance (Lipton et al. [62]). Pohlen et al. [63] build on distributed prioritized replay (ApeX) and incorporate an imitation loss, as well as sampling from expert demonstrations to overcome exploration in sparse reward games of Atari. Kickstarting Deep Reinforcement Learning (Schmitt et al. [64]) learns from already trained teacher agents for faster initial performance.

Compared to standard DRL algorithms, combining DRL and imitation learning is able to improve initial learning performance. Compared to pure imitation learning they are able to improve upon the expert policies.

### 4.2.1 Deep Q-Learning from Demonstrations

Deep $Q$-Learning from Demonstrations (Hester et al. [59]) uses two techniques to speed up training with demonstration data.

First it fills up the replay buffer with demonstrations at the beginning of training. However, this alone would be poor use of the demonstrations as they are quickly overwritten with inferior transitions sampled by the agent. To address this issue, demonstrations are kept permanently in the replay buffer and sampled using prioritized replay. This first technique learns a self-consistent $Q$-value function by reducing the typical $Q$-value loss. The second way to take advantage of the demonstration samples is by imitating the demonstrated policy. The learning updates are aided by a supervised loss which compares the trained policy to an expert policy:

$$L_E(Q) = \max_{a \in \mathcal{A}}(Q(s,a) + l(a_e, a)) - Q(s, a_E)$$

Here $a_E$ is the action taken in the expert demonstrations. $l(a_e, a)$ is zero for $a_e = a$ and positive otherwise. The initial training phase of the $Q$-value function has to focus on the few states covered by the demonstration data. Adding this loss ensures that actions not covered in the demonstrations stay at reasonable values. The complete algorithm has two phases. In the first phase the initial $Q$-value function is trained without any interaction with the environment solely sampling from expert demonstration data. In a second phase the agent trains like the baseline agent, but permanently keeps the demonstration data in the replay buffer. This way the policy, when the agent starts interacting with the environment, is already much better than random.

The conducted experiments show that performance is better in 41 of 42 Atari games in the initial 1 million frames. For final overall performance, the method improves on some games and reaches state-of-the-art scores, including some hard exploration games. In these games the demonstrations can replace exploration bonuses.

### 4.2.2 Kickstarting Deep Reinforcement Learning

Another way to improve initial performance is presented in Kickstarting Deep Reinforcement Learning (Schmitt et al. [64]). The background of this method is different in that it assumes a previous DRL agent is already trained. In this setting the goal is to quickly learn from the teacher policy and then improve on it as the teacher policy becomes less helpful over the course of training.

The teacher policy is used to improve the policy of a student by encouraging the student to match the teacher's actions using an auxiliary loss. Note how the teacher's policy $\pi_T$ needs to be evaluated in the same state as the student's $\pi_S$.

$$L_{distill} = D_{KL}(\pi_T(a|x_t)||\pi_S(a|x_t, \phi))$$

The difference between these policies is measured on the trajectories created by the student, so the student alone can explore parts of the state space the teacher did not visit. This loss $L_{distill}$ is than weighted against the standard DRL loss. During training the weight is slowly reduced so the student becomes independent of the teacher.

Chapter 5

# The Arcade Learning Environment

The Atari 2600 is a home video game console first sold in 1977. Games include classic arcade games ported to the console like *Pac-Man* or *Space Invader*. In these arcade games actions such as eating dots or shooting down aliens awards points and the goal is to reach a high score. Through time games became more complex with first action-adventure games such as *Pitfall!* or later *Montezuma's Revenge*. These games retain an arcade like score, but points are rewarded only rarely, usually for completing subgoals such as collecting keys. The Arcade Learning Environment (Bellemare et al. [65]) provides an interface to Atari 2600 games for reinforcement learning algorithms.

Atari 2600 games are rendered on $120 \times 180$ 7-bit color image. See Figure 5.1 for screenshots. The controls consist of a joystick for 8 directions and a paddle with a single button. Factoring in in no-ops for both direction and paddle this yields 18 possible actions. Though effectively the action space can be as small as 3 in games like *Pong*.

The default benchmark based on the Arcade Learning Environment consists of 57 games. These games offer a variety of game mechanics and visual inputs. They offer challenges to representation learning, planning and exploration making them a suitable benchmark to test generalization for DRL agents. To test the generalization of an agent, its hyperparameters are tuned on a small subset of games. After that, the goal is to solve all games without any fine-tuning to them individually. Solving a game means reaching a high-score comparable to human play. Despite steady progress from the first DRL algorithms, some games like *Montezuma's Revenge* remain unsolved. Table 5.1 provides a rough taxonomy of the Atari games based on the baseline performance of the deep $Q$-learning algorithm and how well it compares to human play.

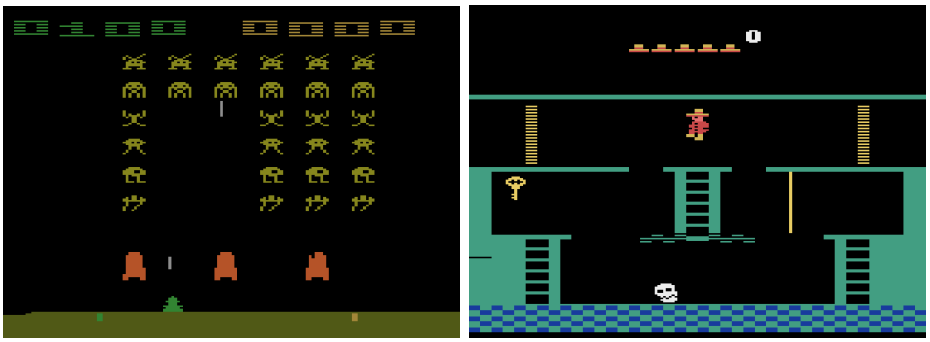

Figure 5.1: Screenshots of two Atari Games: *Space Invaders* (left) and *Montezuma's Revenge* (right). *Space Invader* is considered an easy exploration game as it has dense rewards guiding an agent to a simple and successful control policy. Meanwhile *Montezuma's Revenge* is one of the most challenging games. It requires planning over multiple time-steps to collect keys and discover new rooms.

| Easy Exploration | | | Hard Exploration | |
|---|---|---|---|---|
| Human-Optimal | | Score Exploit | Dense Reward | Sparse Reward |
| Assault | Asterix | Beam Rider | Alien | Freeway |
| Asteroids | Atlantis | Kangaroo | Amidar | Gravitar |
| Battle Zone | Berzerk | Krull | Bank Heist | Montezuma's Revenge |
| Bowling | Boxing | Kung-fu Master | Frostbite | Pitfall! |
| Breakout | Centipede | Road Runner | H.E.R.O. | Private Eye |
| Chopper Cmd | Crazy Climber | Seaquest | Ms. Pac-Man | Solaris |
| Defender | Demon Attack | Up n Down | Q*Bert | Venture |
| Double Dunk | Enduro | Tutankham | Surround | |
| Fishing Derby | Gopher | | Wizard of Wor | |
| Ice Hockey | James Bond | | Zaxxon | |
| Name this Game | Phoenix | | | |
| Pong | River Raid | | | |
| Robotank | Skiing | | | |
| Space Invaders | Stargunner | | | |

Table 5.1: A taxonomy of Atari 2600 games based on exploration difficulty presented in [17]. Easy exploration are games where a DQN agent with $\epsilon$-greedy exploration is sufficient for a high scoring policy. These are subdivided into games where RL agents find a score exploit and reach high-scores without reaching a game's objective. Hard Exploration Games are divided depending on reward frequency.

## 5.1 Atari Benchmarks

Despite the prominence of the Atari benchmarking suite to evaluate recent DRL algorithms, it remains challenging to compare performance between them. The difference between game scores can have various sources apart from the developed exploration or exploitation method. Differences include:

- **Starting Regime:** The no-op regime inserts up to 30 no-op actions at the beginning of an episode. Human starts sample from 100 human starts for the first 30 frames and thus require a more robust policy.
- **Terminal Loss of Life:** Most benchmarks terminate an episode on loss of life. Letting an agent continue playing with the remaining lives improves scores in unforgiving games like Montezuma's Revenge.
- **Episode Length:** Episodes are always limited in time. Most commonly 30 minutes, $108K$ frames, but evaluation time can as low as $18K$ frames.
- **Preprocessing:** Frames are usually stacked 4 at a time, down sampled and gray-scaled. Yielding a $84 \times 84 \times 4$ tensor as state input. The reward signal is also preprocessed by clipping it to the range $[-1, 1]$ for stability reasons.

Aside from differences in Atari benchmark configuration, different base agents also have a considerable influence on the performance of a final agent. Extensions to DQN like prioritized experience replay, the multi-step update rule, dueling architecture and more all address different problems in DRL. The resulting performance increase is complementary to that of an exploration method [27]. So while a specialized method for exploration may show good improvement over their baseline, it may stack up badly compared to more advanced exploration methods. Training time may also be a deciding factor for final performance. While most commonly every game is trained for $200M$ frames, the distributed prioritized experience replay agent (ApeX) uses up to $228000M$. Meanwhile methods focused on initial performance like episodic control only use $10M$ frames training time.

## 5.2 Comparison

Despite all the differences, presented below are some comparisons between the different methods summarized in this paper. First a comparison on performance off the full benchmark. Then on the selection of hard exploration games that has been the focus of most research. Compared is the human normalized score

defined as in [3]:

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}$$

Here the human score is from an average player as in [4]. An agent acting uniformly at random at every step is evaluated for the baseline. The normalized score allows us to compare performance across games. Figure 5.2 shows the median score across all games for both human starts and random starts. In Figure 5.3 the selection is restricted to the hard exploration games with sparse rewards, a focus of much of the research. Lastly Figure 5.4 presents the scores on *Montezuma's Revenge*, one of the most challenging games and thus the most benchmarked game. All scores are taken from the corresponding publications. Where a full benchmark was not available originally, a rerun from a later publication with more games scored is shown.

The benchmarks show agents achieve super-human performance across a majority of games, but the suite of 57 Atari games remains a challenge overall. A focus on only sparse reward games reveals that sophisticated methods for exploration or distributed methods are required for some progress, while agents learning to imitate human play from demonstration show the best performance. These benchmarks should however not be taken as a measure for exploration effectiveness of the individual agents, as many do not aim for overall performance, but rather improvement over some baseline. As an example, consider *DQN Noisynet* vs. *DDQN Dueling Noisynet* in Figure 5.3. While the exploration method is shared, exchanging the base agent DQN with DDQN and the dueling target, yields far superior performance.

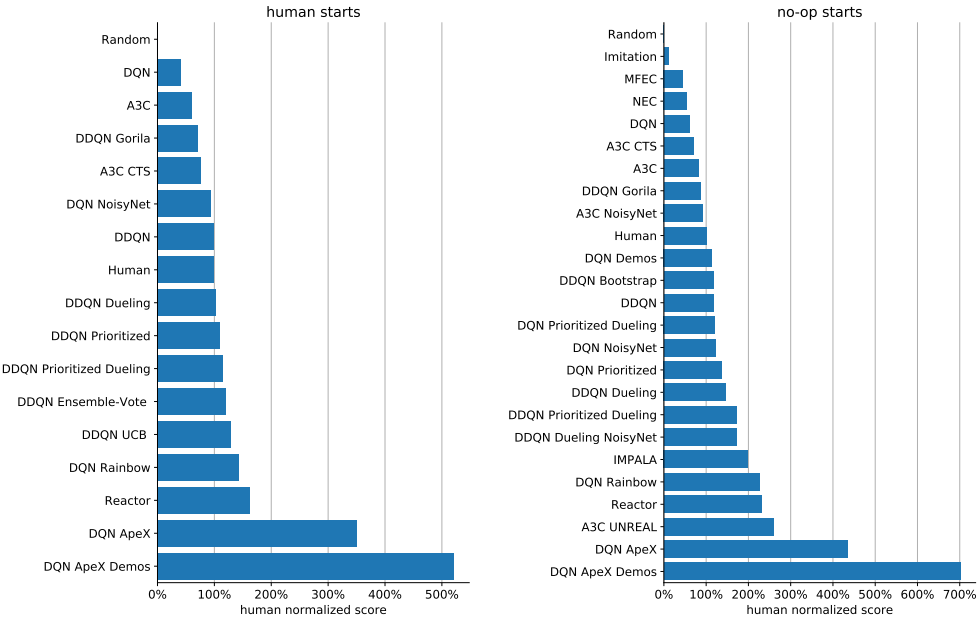


Figure 5.2: Median score across all games. Both for the human starts (left) and random starts (right) a variety of methods is able to surpass median human performance.

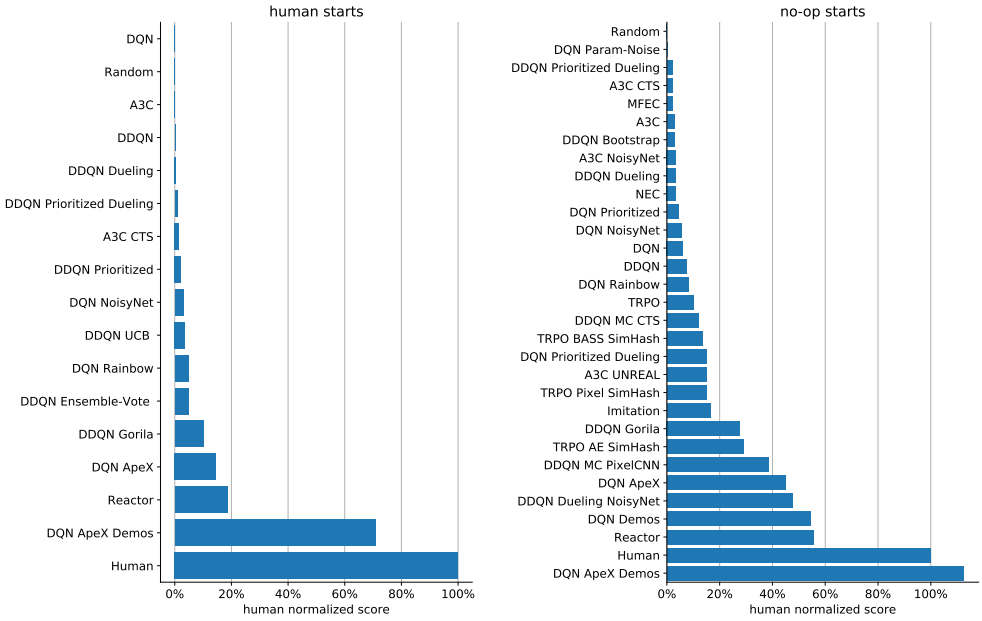


Figure 5.3: Median score for sparse reward games only. This selection offers a bigger challenge to the exploration capability. The more difficult benchmark with human starts (left) sees no method surpass human performance. While in the no-op starts regime (right) only methods utilizing demonstrations are able.

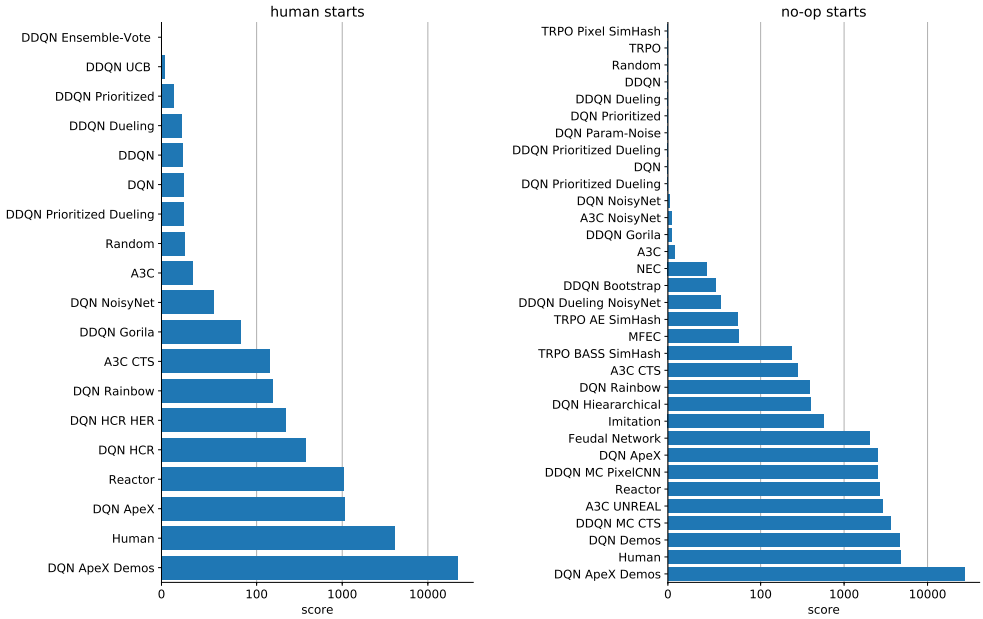


Figure 5.4: Raw scores for *Montezuma's Revenge*. As one of the hardest games many trained agents fail to get past the first room, which would yield 400 points.

Chapter 6

# Conclusion

Presented is an overview of a diverse set of exploration and exploitation methods applicable to deep reinforcement learning problems. There exist different types of exploration, from intrinsic motivation methods gathering information about the environment dynamics, to count-based methods inspired by the optimism in the face of uncertainty principle. Exploring in parameter space, rather than state-action space leads to deep exploration policies. In curriculum learning and hierarchical learning, tasks are split into smaller, easier to solve subtasks, enabling efficient and guided exploration. Distributed architecture make use of parallel compute resources to enable faster learning. More efficient exploitation is demonstrated in episodic memory based methods, aiming to learn from single experiences. In the learning from expert demonstration settings, agents copy from demonstrations for initial performance and then learn to surpass them.

All these approaches take inspiration from human learning, tabular RL or other areas of machine learning. While each shows promise in its own regard, it has proven difficult to compare performance between them. Nevertheless, the Atari benchmarks show overall progress of DRL in hard exploration problems.

# Bibliography


[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518** (Feb 2015) 529 EP –

[2] Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: A brief survey of deep reinforcement learning. CoRR **abs/1708.05866** (2017)

[3] van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. CoRR **abs/1509.06461** (2015)

[4] Wang, Z., de Freitas, N., Lanctot, M.: Dueling network architectures for deep reinforcement learning. CoRR **abs/1511.06581** (2015)

[5] Ostrovski, G., Bellemare, M.G., van den Oord, A., Munos, R.: Count-based exploration with neural density models. CoRR **abs/1703.01310** (2017)

[6] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. CoRR **abs/1511.05952** (2015)

[7] Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. CoRR **abs/1602.01783** (2016)

[8] Chapelle, O., Li, L.: An empirical evaluation of thompson sampling. In Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q., eds.: Advances in Neural Information Processing Systems 24. Curran Associates, Inc. (2011) 2249–2257

[9] Stadie, B.C., Levine, S., Abbeel, P.: Incentivizing exploration in reinforcement learning with deep predictive models. CoRR **abs/1507.00814** (2015)

[10] Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. CoRR **abs/1705.05363** (2017)

[11] Haber, N., Mrowca, D., Fei-Fei, L., Yamins, D.L.K.: Learning to play with intrinsically-motivated self-aware agents. CoRR **abs/1802.07442** (2018)

[12] de Abril, I.M., Kanai, R.: Curiosity-driven reinforcement learning with homeostatic regulation. CoRR **abs/1801.07440** (2018)

[13] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., Abbeel, P.: VIME: Variational Information Maximizing Exploration. ArXiv e-prints (May 2016)

[14] Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement Learning with Unsupervised Auxiliary Tasks. ArXiv e-prints (November 2016)

[15] Shelhamer, E., Mahmoudieh, P., Argus, M., Darrell, T.: Loss is its own reward: Self-supervision for reinforcement learning. CoRR **abs/1612.07307** (2016)

[16] Strehl, A.L., Littman, M.L.: An analysis of model-based interval estimation for markov decision processes. Journal of Computer and System Sciences **74**(8) (2008) 1309 – 1331 Learning Theory 2005.

[17] Bellemare, M.G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. CoRR **abs/1606.01868** (2016)

[18] Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F.D., Abbeel, P.: #exploration: A study of count-based exploration for deep reinforcement learning. CoRR **abs/1611.04717** (2016)

[19] Martin, J., Sasikumar, S.N., Everitt, T., Hutter, M.: Count-based exploration in feature space for reinforcement learning. CoRR **abs/1706.08090** (2017)

[20] Machado, M.C., Bellemare, M.G., Bowling, M.: Count-based exploration with the successor representation. arXiv preprint arXiv:1807.11622 (2018)

[21] Bellemare, M., Veness, J., Talvitie, E.: Skip context tree switching. In: International Conference on Machine Learning. (2014) 1458–1466

[22] Osband, I., Blundell, C., Pritzel, A., Roy, B.V.: Deep exploration via bootstrapped DQN. CoRR **abs/1602.04621** (2016)

[23] Chen, R.Y., Sidor, S., Abbeel, P., Schulman, J.: Ucb exploration via q-ensembles. CoRR **abs/1706.01502** (2017)

[24] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M.: Parameter space noise for exploration. CoRR **abs/1706.01905** (2017)

[25] Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)

[26] Fortunato, M., Azar, M.G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., Legg, S.: Noisy networks for exploration. CoRR **abs/1706.10295** (2017)

[27] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M.G., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. CoRR **abs/1710.02298** (2017)

[28] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A.D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., Silver, D.: Massively parallel methods for deep reinforcement learning. CoRR **abs/1507.04296** (2015)

[29] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., Silver, D.: Distributed prioritized experience replay. CoRR **abs/1803.00933** (2018)

[30] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz, J.: GA3C: gpu-based A3C for deep reinforcement learning. CoRR **abs/1611.06256** (2016)

[31] Clemente, A.V., Martínez, H.N.C., Chandra, A.: Efficient parallel methods for deep reinforcement learning. CoRR **abs/1705.04862** (2017)

[32] Gruslys, A., Azar, M.G., Bellemare, M.G., Munos, R.: The reactor: A sample-efficient actor-critic architecture. CoRR **abs/1704.04651** (2017)

[33] Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M.G.: Safe and efficient off-policy reinforcement learning. CoRR **abs/1606.02647** (2016)

[34] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., Kavukcuoglu, K.: IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. CoRR **abs/1802.01561** (2018)

[35] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning (2009)

[36] Schaul, T., Horgan, D., Gregor, K., Silver, D.: Universal value function approximators. In Bach, F., Blei, D., eds.: Proceedings of the 32nd International Conference on Machine Learning. Volume 37 of Proceedings of Machine Learning Research., Lille, France, PMLR (07–09 Jul 2015) 1312–1320

[37] Sukhbaatar, S., Kostrikov, I., Szlam, A., Fergus, R.: Intrinsic motivation and automatic curricula via asymmetric self-play. CoRR **abs/1703.05407** (2017)

[38] Florensa, C., Held, D., Wulfmeier, M., Abbeel, P.: Reverse curriculum generation for reinforcement learning. CoRR **abs/1707.05300** (2017)

[39] Florensa, C., Held, D., Geng, X., Abbeel, P.: Automatic goal generation for reinforcement learning agents. In: International Conference on Machine Learning. (2018) 1514–1523

[40] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. (2014) 2672–2680

[41] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W.: Hindsight experience replay. CoRR **abs/1707.01495** (2017)

[42] Molchanov, A., Hausman, K., Birchfield, S., Sukhatme, G.: Region Growing Curriculum Generation for Reinforcement Learning. ArXiv e-prints (July 2018)

[43] Czarnecki, W.M., Jayakumar, S.M., Jaderberg, M., Hasenclever, L., Teh, Y.W., Osindero, S., Heess, N., Pascanu, R.: Mix&match-agent curricula for reinforcement learning. arXiv preprint arXiv:1806.01780 (2018)

[44] Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence **112**(1-2) (1999) 181–211

[45] Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.B.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. CoRR **abs/1604.06057** (2016)

[46] Bacon, P., Harb, J., Precup, D.: The option-critic architecture. CoRR **abs/1609.05140** (2016)

[47] Vezhnevets, A.S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., Kavukcuoglu, K.: Feudal networks for hierarchical reinforcement learning. CoRR **abs/1703.01161** (2017)

[48] Harb, J., Bacon, P., Klissarov, M., Precup, D.: When waiting is not an option : Learning options with a deliberation cost. CoRR **abs/1709.04571** (2017)

[49] Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J.Z., Rae, J.W., Wierstra, D., Hassabis, D.: Model-free episodic control. CoRR **abs/1606.04460** (2016)

[50] Brea, J.: Is prioritized sweeping the better episodic control? CoRR **abs/1711.06677** (2017)

[51] Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)

[52] Pritzel, A., Uria, B., Srinivasan, S., Badia, A.P., Vinyals, O., Hassabis, D., Wierstra, D., Blundell, C.: Neural episodic control. CoRR **abs/1703.01988** (2017)

[53] Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K.: End to end learning for self-driving cars. CoRR **abs/1604.07316** (2016)

[54] Ho, J., Gupta, J.K., Ermon, S.: Model-free imitation learning with policy optimization. CoRR **abs/1605.08478** (2016)

[55] Ho, J., Ermon, S.: Generative adversarial imitation learning. CoRR **abs/1606.03476** (2016)

[56] Ross, S., Gordon, G., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. (2011) 627–635

[57] Sun, W., Venkatraman, A., Gordon, G.J., Boots, B., Bagnell, J.A.: Deeply aggrevated: Differentiable imitation learning for sequential prediction. CoRR **abs/1703.01030** (2017)

[58] Hosu, I., Rebedea, T.: Playing atari games with deep reinforcement learning and human checkpoint replay. CoRR **abs/1607.05077** (2016)

[59] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J.Z., Gruslys, A.: Deep Q-learning from Demonstrations. ArXiv e-prints (April 2017)

[60] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., Riedmiller, M.A.: Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. CoRR **abs/1707.08817** (2017)

[61] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., Abbeel, P.: Overcoming exploration in reinforcement learning with demonstrations. CoRR **abs/1709.10089** (2017)

[62] Lipton, Z.C., Gao, J., Li, L., Li, X., Ahmed, F., Deng, L.: Efficient exploration for dialog policy learning with deep BBQ networks \& replay buffer spiking. CoRR **abs/1608.05081** (2016)

[63] Pohlen, T., Piot, B., Hester, T., Azar, M.G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Vecerík, M., Hessel, M., Munos, R., Pietquin, O.: Observe and look further: Achieving consistent performance on atari. CoRR **abs/1805.11593** (2018)

[64] Schmitt, S., Hudson, J.J., Zídek, A., Osindero, S., Doersch, C., Czarnecki, W.M., Leibo, J.Z., Küttler, H., Zisserman, A., Simonyan, K., Eslami, S.M.A.: Kickstarting deep reinforcement learning. CoRR **abs/1803.03835** (2018)

[65] Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. CoRR **abs/1207.4708** (2012)